Bazaar User Reference

Release 2.8.0dev1

Bazaar Developers

March 22, 2019

CONTENTS

CHAPTER

ONE

ABOUT THIS MANUAL

This manual is generated from Bazaar's online help. To use the online help system, try the following commands.

Introduction including a list of commonly used commands:

```
bzr help
```

List of topics and a summary of each:

```
bzr help topics
```

List of commands and a summary of each:

```
bzr help commands
```

More information about a particular topic or command:

```
bzr help topic-or-command-name
```

The following web sites provide further information on Bazaar:

Home page http://bazaar.canonical.com/

Official docs http://doc.bazaar.canonical.com/

Launchpad https://launchpad.net/bzr/

CHAPTER

TWO

CONCEPTS

2.1 branches

Purpose List the branches available at the current location.

Usage bzr branches [LOCATION]

Options

--usage	Show usage message and options.
-R, --recursive	Recursively scan for branches rather than just looking in the specified location.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description This command will print the names of all the branches at the current location.

2.2 Checkouts

Checkouts are source trees that are connected to a branch, so that when you commit in the source tree, the commit goes into that branch. They allow you to use a simpler, more centralized workflow, ignoring some of Bazaar's decentralized features until you want them. Using checkouts with shared repositories is very similar to working with SVN or CVS, but doesn't have the same restrictions. And using checkouts still allows others working on the project to use whatever workflow they like.

A checkout is created with the bzr checkout command (see "help checkout"). You pass it a reference to another branch, and it will create a local copy for you that still contains a reference to the branch you created the checkout from (the master branch). Then if you make any commits they will be made on the other branch first. This creates an instant mirror of your work, or facilitates lockstep development, where each developer is working together, continuously integrating the changes of others.

However the checkout is still a first class branch in Bazaar terms, so that you have the full history locally. As you have a first class branch you can also commit locally if you want, for instance due to the temporary loss af a network connection. Use the –local option to commit to do this. All the local commits will then be made on the master branch the next time you do a non-local commit.

If you are using a checkout from a shared branch you will periodically want to pull in all the changes made by others. This is done using the "update" command. The changes need to be applied before any non-local commit, but Bazaar will tell you if there are any changes and suggest that you use this command when needed.

It is also possible to create a "lightweight" checkout by passing the –lightweight flag to checkout. A lightweight checkout is even closer to an SVN checkout in that it is not a first class branch, it mainly consists of the working tree. This means that any history operations must query the master branch, which could be slow if a network connection is involved. Also, as you don't have a local branch, then you cannot commit locally.

Lightweight checkouts work best when you have fast reliable access to the master branch. This means that if the master branch is on the same disk or LAN a lightweight checkout will be faster than a heavyweight one for any commands that modify the revision history (as only one copy of the branch needs to be updated). Heavyweight checkouts will generally be faster for any command that uses the history but does not change it, but if the master branch is on the same disk then there won't be a noticeable difference.

Another possible use for a checkout is to use it with a treeless repository containing your branches, where you maintain only one working tree by switching the master branch that the checkout points to when you want to work on a different branch.

Obviously to commit on a checkout you need to be able to write to the master branch. This means that the master branch must be accessible over a writeable protocol , such as sftp://, and that you have write permissions at the other end. Checkouts also work on the local file system, so that all that matters is file permissions.

You can change the master of a checkout by using the "switch" command (see "help switch"). This will change the location that the commits are sent to. The "bind" command can also be used to turn a normal branch into a heavy checkout. If you would like to convert your heavy checkout into a normal branch so that every commit is local, you can use the "unbind" command. To see whether or not a branch is bound or not you can use the "info" command. If the branch is bound it will tell you the location of the bound branch.

Related commands:

```
checkout    Create a checkout. Pass --lightweight to get a lightweight
            checkout
update      Pull any changes in the master branch in to your checkout
commit      Make a commit that is sent to the master branch. If you have
            a heavy checkout then the --local option will commit to the
            checkout without sending the commit to the master
switch      Change the master branch that the commits in the checkout will
            be sent to
bind        Turn a standalone branch into a heavy checkout so that any
            commits will be sent to the master branch
unbind      Turn a heavy checkout into a standalone branch so that any
            commits are only made locally
info        Displays whether a branch is bound or unbound. If the branch is
            bound, then it will also display the location of the bound branch
```

2.3 Content Filters

2.3.1 Content formats

Bazaar's content filtering allows you to store files in a different format from the copy in your working tree. This lets you, or your co-developers, use Windows development tools that expect CRLF files on projects that use other line-ending conventions. Among other things, content filters also let Unix developers more easily work on projects using Windows line-ending conventions, keyword expansion/compression, and trailing spaces on lines in text files to be implicitly stripped when committed.

To generalize, there are two content formats supported by Bazaar:

- a canonical format - how files are stored internally
- a convenience format - how files are created in a working tree.

2.3.2 Format conversion

The conversion between these formats is done by content filters. A content filter has two parts:

- a read converter - converts from convenience to canonical format
- a write converter - converts from canonical to convenience format.

Many of these converters will provide *round-trip* conversion, i.e. applying the read converter followed by the write converter gives back the original content. However, others may provide an asymmetric conversion. For example, a read converter might strip trailing whitespace off lines in source code while the matching write converter might pass content through unchanged.

2.3.3 Enabling content filters

Content filters are typically provided by plugins, so the first step in using them is to install the relevant plugins and read their documentation. Some plugins may be very specific about which files they filter, e.g. only files ending in `.java` or `.php`. In other cases, the plugin may leave it in the user's hands to define which files are to be filtered. This is typically done using rule-based preferences. See `bzr help rules` for general information about defining these.

2.3.4 Impact on commands

Read converters are only applied to commands that read content from a working tree, e.g. status, diff and commit. For example, `bzr diff` will apply read converters to files in the working tree, then compare the results to the content last committed.

Write converters are only applied by commands that **create files in a working tree**, e.g. branch, checkout, update. If you wish to see the canonical format of a file or tree, use `bzr cat` or `bzr export` respectively.

Note: `bzr commit` does not implicitly apply write converters after comitting files. If this makes sense for a given plugin providing a content filter, the plugin can usually achieve this effect by using a `start_commit` or `post_commit` hook say. See *Hooks* for more information on hooks.

2.3.5 Refreshing your working tree

For performance reasons, Bazaar caches the timestamps of files in a working tree, and assumes files are unchanged if their timestamps match the cached values. As a consequence, there are times when you may need to explicitly ask for content filtering to be reapplied in one or both directions, e.g. after installing or reconfiguring plugins providing it.

Here are some general guidelines for doing this:

- To reapply read converters, `touch` files, i.e. update their timestamp. Operations like `bzr status` should then reapply the relevant read converters and compare the end result with the canonical format.
- To reapply write converters, ensure there are no local changes, delete the relevant files and run `bzr revert` on those files.

Note: In the future, it is likely that additional options will be added to commands to make this refreshing process faster and safer.

2.4 Criss-Cross

A criss-cross in the branch history can cause the default merge technique to emit more conflicts than would normally be expected.

In complex merge cases, `bzr merge --lca` or `bzr merge --weave` may give better results. You may wish to `bzr revert` the working tree and merge again. Alternatively, use `bzr remerge` on particular conflicted files.

Criss-crosses occur in a branch's history if two branches merge the same thing and then merge one another, or if two branches merge one another at the same time. They can be avoided by having each branch only merge from or into a designated central branch (a "star topology").

Criss-crosses cause problems because of the way merge works. Bazaar's default merge is a three-way merger; in order to merge OTHER into THIS, it must find a basis for comparison, BASE. Using BASE, it can determine whether differences between THIS and OTHER are due to one side adding lines, or from another side removing lines.

Criss-crosses mean there is no good choice for a base. Selecting the recent merge points could cause one side's changes to be silently discarded. Selecting older merge points (which Bazaar does) mean that extra conflicts are emitted.

The `weave` merge type is not affected by this problem because it uses line-origin detection instead of a basis revision to determine the cause of differences.

2.5 Diverged Branches

When Bazaar tries to push one branch onto another, it requires that the destination branch must be ready to receive the source branch. If this isn't the case, then we say that the branches have `diverged`. Branches are considered diverged if the destination branch's most recent commit is one that has not been merged (directly or indirectly) by the source branch. To recover from diverged branches, one must merge the missing revisions into the source branch.

This situation commonly arises when using a centralized workflow with local commits. If someone else has committed new work to the mainline since your last pull and you have local commits that have not yet been pushed to the mainline, then your local branch and the mainline have diverged.

2.5.1 Discovering What Has Diverged

The `bzr missing` command is used to find out what revisions are in another branch that are not present in the current branch, and vice-versa. It shows a summary of which extra revisions exist in each branch. If you want to see the precise effects of those revisions, you can use `bzr diff --old=other_branch` to show the differences between other_branch and your current branch.

2.5.2 A Solution

The solution is to merge the revisions from the mainline into your local branch. To do so, use `bzr merge` to get the new revisions from the mainline. This merge may result in conflicts if the other developer's changes overlap with your changes. These conflicts should be resolved before continuing. After any conflicts have been resolved, or even if there were no conflicts, Bazaar requires that you explicitly commit these new revisions to your local branch. This requirement gives you an opportunity to test the resulting working tree for correctness, since the merged revisions could have made arbitrary changes. After testing, you should commit the merge using `bzr commit`. This clears up the diverged branches situation. Your local branch can now be pushed to the mainline.

2.6 End of Line Conversion

EOL conversion is provided as a content filter where Bazaar internally stores a canonical format but outputs a convenience format. See `bzr help content-filters` for general information about using these.

Note: Content filtering is only supported in recently added formats, e.g. 1.14. Be sure that both the repository *and* the branch are in a recent format. (Just setting the format on the repository is not enough.) If content filtering does not appear to be working, use 'bzr info -v' to confirm that the branch is using "Working tree format 5" or later.

EOL conversion needs to be enabled for selected file patterns using rules. See `bzr help rules` for general information on defining rules. Currently, rules are only supported in $BZR_HOME/.bazaar/rules (or %BZR_HOME%/bazaar/2.0/rules on Windows). Branch specific rules will be supported in a future verison of Bazaar.

To configure which files to filter, set `eol` to one of the values below. (If a value is not set, `exact` is the default.)

Value	Checkout end-of-lines as	Commit end-of-lines as
`native`	`crlf` on Windows, `lf` otherwise	`lf`
`lf`	`lf`	`lf`
`crlf`	`crlf`	`lf`
`exact`	No conversion	Exactly as in file

Note: For safety reasons, no conversion is applied to any file where a null byte is detected in the file.

For users working on a cross-platform project, here is a suggested rule to use as a starting point:

```
[name *]
eol = native
```

If you have binary files that do not contain a null byte though, be sure to add `eol = exact` rules for those as well. You can do this by giving more explicit patterns earlier in the rules file. For example:

```
[name *.png]
eol = exact

[name *]
eol = native
```

If your working tree is on a network drive shared by users on different operating systems, you typically want to force certain conventions for certain files. In that way, if a file is created with the wrong line endings or line endings get mixed during editing, it gets committed correctly and gets checked out correctly. For example:

```
[name *.bat]
eol = crlf

[name *.sh]
eol = lf

[name *]
eol = native
```

If you take the care to create files with their required endings, you can achieve *almost* the same thing by using `eol = exact`. It is slightly safer to use `lf` and `crlf` though because edits accidentally introducing mixed line endings will be corrected during commit for files with those settings.

If you have sample test data that deliberately has text files with mixed newline conventions, you can ask for those to be left alone like this:

```
[name test_data/]
eol = exact

[name *]
eol = native
```

Note that `exact` does not imply the file is binary but it does mean that no conversion of end-of-lines will be done. (Bazaar currently relies of content analysis to detect binary files for commands like `diff`. In the future, a `binary = true` rule may be added but it is not supported yet.)

If you have an existing repository with text files already stored using Windows newline conventions (`crlf`), then you may want to keep using that convention in the repository. Forcing certain files to this convention may also help users who do not have rules configured. To do this, set `eol` to one of the values below.

Value	Checkout end-of-lines as	Commit end-of-lines as
`native-with-crlf-in-repo`	`crlf` on Windows, `lf` otherwise	`crlf`
`lf-with-crlf-in-repo`	`lf`	`crlf`
`crlf-with-crlf-in-repo`	`crlf`	`crlf`

For users working on an existing project that uses Windows newline conventions in their Bazaar repository, this rule is suggested as a starting point:

```
[name *]
eol = native-with-crlf-in-repo
```

For new projects, it is recommended that end-of-lines be stored as `lf` and that users stick to the basic settings, i.e. `native`, `lf`, `crlf` and `exact`.

Note: Bazaar's EOL conversion will convert the content of files but never reject files because a given line ending or mixed line endings are found. A precommit hook should be used if you wish to validate (and not just convert) content before committing.

2.7 Storage Formats

To ensure that older clients do not access data incorrectly, Bazaar's policy is to introduce a new storage format whenever new features requiring new metadata are added. New storage formats may also be introduced to improve performance and scalability.

The newest format, 2a, is highly recommended. If your project is not using 2a, then you should suggest to the project owner to upgrade.

Note: Some of the older formats have two variants: a plain one and a rich-root one. The latter include an additional field about the root of the tree. There is no performance cost for using a rich-root format but you cannot easily merge changes from a rich-root format into a plain format. As a consequence, moving a project to a rich-root format takes some co-ordination in that all contributors need to upgrade their repositories around the same time. 2a and all future formats will be implicitly rich-root.

See *Current Storage Formats* for the complete list of currently supported formats. See *Other Storage Formats* for descriptions of any available experimental and deprecated formats.

2.8 Patterns

Bazaar uses patterns to match files at various times. For example, the `add` command skips over files that match ignore patterns and preferences can be associated with files using rule patterns. The pattern syntax is described below.

Trailing slashes on patterns are ignored. If the pattern contains a slash or is a regular expression, it is compared to the whole path from the branch root. Otherwise, it is compared to only the last component of the path. To match a file only in the root directory, prepend `./`. Patterns specifying absolute paths are not allowed.

Patterns may include globbing wildcards such as:

```
? - Matches any single character except '/'
* - Matches 0 or more characters except '/'
```

```
/**/ - Matches 0 or more directories in a path
[a-z] - Matches a single character from within a group of characters
```

Patterns may also be Python regular expressions. Regular expression patterns are identified by a `RE:` prefix followed by the regular expression. Regular expression patterns may not include named or numbered groups.

Case insensitive ignore patterns can be specified with regular expressions by using the `i` (for ignore case) flag in the pattern.

For example, a case insensitive match for `foo` may be specified as:

```
RE:(?i)foo
```

Ignore patterns may be prefixed with `!`, which means that a filename matched by that pattern will not be ignored.

2.9 Repositories

Repositories in Bazaar are where committed information is stored. There is a repository associated with every branch.

Repositories are a form of database. Bzr will usually maintain this for good performance automatically, but in some situations (e.g. when doing very many commits in a short time period) you may want to ask bzr to optimise the database indices. This can be done by the 'bzr pack' command.

By default just running 'bzr init' will create a repository within the new branch but it is possible to create a shared repository which allows multiple branches to share their information in the same location. When a new branch is created it will first look to see if there is a containing shared repository it can use.

When two branches of the same project share a repository, there is generally a large space saving. For some operations (e.g. branching within the repository) this translates in to a large time saving.

To create a shared repository use the init-repository command (or the alias init-repo). This command takes the location of the repository to create. This means that 'bzr init-repository repo' will create a directory named 'repo', which contains a shared repository. Any new branches that are created in this directory will then use it for storage.

It is a good idea to create a repository whenever you might create more than one branch of a project. This is true for both working areas where you are doing the development, and any server areas that you use for hosting projects. In the latter case, it is common to want branches without working trees. Since the files in the branch will not be edited directly there is no need to use up disk space for a working tree. To create a repository in which the branches will not have working trees pass the '–no-trees' option to 'init-repository'.

Related commands:

```
init-repository   Create a shared repository. Use --no-trees to create one
                  in which new branches won't get a working tree.
```

2.10 Rules

2.10.1 Introduction

Rules are defined in ini file format where the sections are file glob patterns and the contents of each section are the preferences for files matching that pattern(s). For example:

```
[name *.bat]
eol = native
```

```
[name *.html *.xml]
keywords = xml_escape
```

Preferences like these are useful for commands and plugins wishing to provide custom behaviour for selected files. For more information on end of line conversion see *End of Line Conversion*. Keyword support is provided by the keywords plugin.

2.10.2 Files

Default rules for all branches are defined in the optional file `BZR_HOME/rules`.

2.10.3 Rule Patterns

Patterns are ordered and searching stops as soon as one matches. As a consequence, more explicit patterns should be placed towards the top of the file. Rule patterns use exactly the same conventions as ignore patterns. See *Patterns* for details.

Note: Patterns containing square brackets or spaces should be surrounded in quotes to ensure they are correctly parsed.

2.11 Standalone Trees

A standalone tree is a working tree with an associated repository. It is an independently usable branch, with no dependencies on any other. Creating a standalone tree (via bzr init) is the quickest way to put an existing project under version control.

Related Commands:

```
init    Make a directory into a versioned branch.
```

2.12 Branches Out of Sync

When reconfiguring a checkout, tree or branch into a lightweight checkout, a local branch must be destroyed. (For checkouts, this is the local branch that serves primarily as a cache.) If the branch-to-be-destroyed does not have the same last revision as the new reference branch for the lightweight checkout, data could be lost, so Bazaar refuses.

How you deal with this depends on *why* the branches are out of sync.

If you have a checkout and have done local commits, you can get back in sync by running "bzr update" (and possibly "bzr commit").

If you have a branch and the remote branch is out-of-date, you can push the local changes using "bzr push". If the local branch is out of date, you can do "bzr pull". If both branches have had changes, you can merge, commit and then push your changes. If you decide that some of the changes aren't useful, you can "push –overwrite" or "pull –overwrite" instead.

2.13 Working Trees

A working tree is the contents of a branch placed on disk so that you can see the files and edit them. The working tree is where you make changes to a branch, and when you commit the current state of the working tree is the snapshot that is recorded in the commit.

When you push a branch to a remote system, a working tree will not be created. If one is already present the files will not be updated. The branch information will be updated and the working tree will be marked as out-of-date. Updating a working tree remotely is difficult, as there may be uncommitted changes or the update may cause content conflicts that are difficult to deal with remotely.

If you have a branch with no working tree you can use the 'checkout' command to create a working tree. If you run 'bzr checkout .' from the branch it will create the working tree. If the branch is updated remotely, you can update the working tree by running 'bzr update' in that directory.

If you have a branch with a working tree that you do not want the 'remove-tree' command will remove the tree if it is safe. This can be done to avoid the warning about the remote working tree not being updated when pushing to the branch. It can also be useful when working with a '–no-trees' repository (see 'bzr help repositories').

If you want to have a working tree on a remote machine that you push to you can either run 'bzr update' in the remote branch after each push, or use some other method to update the tree during the push. There is an 'rspush' plugin that will update the working tree using rsync as well as doing a push. There is also a 'push-and-update' plugin that automates running 'bzr update' via SSH after each push.

Useful commands:

```
checkout     Create a working tree when a branch does not have one.
remove-tree  Removes the working tree from a branch when it is safe to do so.
update       When a working tree is out of sync with its associated branch
             this will update the tree to match the branch.
```

CHAPTER

THREE

LISTS

3.1 Authentication Settings

3.1.1 Intent

Many different authentication policies can be described in the `authentication.conf` file but a particular user should need only a few definitions to cover his needs without having to specify a user and a password for every branch he uses.

The definitions found in this file are used to find the credentials to use for a given url. The same credentials can generally be used for as many branches as possible by grouping their declaration around the remote servers that need them. It's even possible to declare credentials that will be used by different servers.

The intent is to make this file as small as possible to minimize maintenance.

Once the relevant credentials are declared in this file you may use branch urls without embedding passwords (security hazard) or even users (enabling sharing of your urls with others).

Instead of using:

```
bzr branch ftp://joe:secret@host.com/path/to/my/branch
```

you simply use:

```
bzr branch ftp://host.com/path/to/my/branch
```

provided you have created the following `authentication.conf` file:

```
[myprojects]
scheme=ftp
host=host.com
user=joe
password=secret
```

3.1.2 Authentication definitions

There are two kinds of authentication used by the various schemes supported by bzr:

1. user and password

`FTP` needs a (`user`, `password`) to authenticate against a `host` `SFTP` can use either a password or a host key to authenticate. However, ssh agents are a better, more secure solution. So we have chosen to not provide our own less secure method.

2. user, realm and password

`HTTP` and `HTTPS` needs a (`user, realm, password`) to authenticate against a host. But, by using `.htaccess` files, for example, it is possible to define several (`user, realm, password`) for a given `host`. So what is really needed is (`user`, `password`, `host`, `path`). The `realm` is not taken into account in the definitions, but will displayed if bzr prompts you for a password.

`HTTP proxy` can be handled as `HTTP` (or `HTTPS`) by explicitly specifying the appropriate port.

To take all schemes into account, the password will be deduced from a set of authentication definitions (`scheme`, `host`, `port`, `path`, `user`, `password`).

- `scheme`: can be empty (meaning the rest of the definition can be used for any scheme), `SFTP` and `bzr+ssh` should not be used here, `ssh` should be used instead since this is the real scheme regarding authentication,
- `host`: can be empty (to act as a default for any host),
- `port` can be empty (useful when an host provides several servers for the same scheme), only numerical values are allowed, this should be used only when the server uses a port different than the scheme standard port,
- `path`: can be empty (FTP or SFTP will never user it),
- `user`: can be empty (`bzr` will defaults to python's `getpass.get_user()`),
- `password`: can be empty if you prefer to always be prompted for your password.

Multiple definitions can be provided and, for a given URL, bzr will select a (`user` [, `password`]) based on the following rules :

1. the first match wins,
2. empty fields match everything,
3. `scheme` matches even if decorators are used in the requested URL,
4. `host` matches exactly or act as a domain if it starts with '.' (`project.bzr.sf.net` will match `.bzr.sf.net` but `projectbzr.sf.net` will not match `bzr.sf.net`).
5. `port` matches if included in the requested URL (exact matches only)
6. `path` matches if included in the requested URL (and by rule #2 above, empty paths will match any provided path).

3.1.3 File format

The general rules for *configuration files* apply except for the variable policies.

Each section describes an authentication definition.

The section name is an arbitrary string, only the `DEFAULT` value is reserved and should appear as the *last* section.

Each section should define:

- `user`: the login to be used,

Each section could define:

- `host`: the remote server,
- `port`: the port the server is listening,
- `path`: the branch location,
- `password`: the password.

3.1.4 Examples

Personal projects hosted outside

All connections are done with the same `user` (the remote one for which the default bzr one is not appropriate) and the password is always prompted with some exceptions:

```
# Pet projects on hobby.net
[hobby]
host=r.hobby.net
user=jim
password=obvious1234

# Home server
[home]
scheme=https
host=home.net
user=joe
password=1essobV10us

[DEFAULT]
# Our local user is barbaz, on all remote sites we're known as foobar
user=foobar
```

Source hosting provider

In the shp.net (fictitious) domain, each project has its own site:

```
[shpnet domain]
# we use sftp, but ssh is the scheme used for authentication
scheme=ssh
# The leading '.' ensures that 'shp.net' alone doesn't match
host=.shp.net
user=joe
# bzr don't support supplying a password for sftp,
# consider using an ssh agent if you don't want to supply
# a password interactively. (pageant, ssh-agent, etc)
```

HTTPS, SFTP servers and their proxy

At company.com, the server hosting release and integration branches is behind a proxy, and the two branches use different authentication policies:

```
[reference code]
scheme=https
host=dev.company.com
path=/dev
user=user1
password=pass1

# development branches on dev server
[dev]
scheme=ssh # bzr+ssh and sftp are available here
host=dev.company.com
path=/dev/integration
user=user2
```

```
# proxy
[proxy]
scheme=http
host=proxy.company.com
port=3128
user=proxyuser1
password=proxypass1
```

3.1.5 Planned enhancements

The following are not yet implemented but planned as parts of a work in progress:

- add a `password_encoding` field allowing:
  - storing the passwords in various obfuscating encodings (base64 for one),
  - delegate password storage to plugins (.netrc for example).
- update the credentials when the user is prompted for user or password,
- add a `verify_certificates` field for `HTTPS`.

The `password_encoding` and `verify_certificates` fields are recognized but ignored in the actual implementation.

3.2 Bug Tracker Settings

When making a commit, metadata about bugs fixed by that change can be recorded by using the `--fixes` option. For each bug marked as fixed, an entry is included in the 'bugs' revision property stating '<url> <status>'. (The only `status` value currently supported is `fixed`.)

The `--fixes` option allows you to specify a bug tracker and a bug identifier rather than a full URL. This looks like:

```
bzr commit --fixes <tracker>:<id>
```

or:

```
bzr commit --fixes <id>
```

where "<tracker>" is an identifier for the bug tracker, and "<id>" is the identifier for that bug within the bugtracker, usually the bug number. If "<tracker>" is not specified the `bugtracker` set in the branch or global configuration is used.

Bazaar knows about a few bug trackers that have many users. If you use one of these bug trackers then there is no setup required to use this feature, you just need to know the tracker identifier to use. These are the bugtrackers that are built in:

URL	Abbreviation	Example
https://bugs.launchpad.net/	lp	lp:12345
http://bugs.debian.org/	deb	deb:12345
http://bugzilla.gnome.org/	gnome	gnome:12345

For the bug trackers not listed above configuration is required. Support for generating the URLs for any project using Bugzilla or Trac is built in, along with a template mechanism for other bugtrackers with simple URL schemes. If your bug tracker can't be described by one of the schemes described below then you can write a plugin to support it.

If you use Bugzilla or Trac, then you only need to set a configuration variable which contains the base URL of the bug tracker. These options can go into `bazaar.conf`, `branch.conf` or into a branch-specific configuration section in `locations.conf`. You can set up these values for each of the projects you work on.

Note: As you provide a short name for each tracker, you can specify one or more bugs in one or more trackers at commit time if you wish.

3.2.1 Launchpad

Use `bzr commit --fixes lp:2` to record that this commit fixes bug 2.

3.2.2 bugzilla_<tracker>_url

If present, the location of the Bugzilla bug tracker referred to by <tracker>. This option can then be used together with `bzr commit --fixes` to mark bugs in that tracker as being fixed by that commit. For example:

```
bugzilla_squid_url = http://bugs.squid-cache.org
```

would allow `bzr commit --fixes squid:1234` to mark Squid's bug 1234 as fixed.

3.2.3 trac_<tracker>_url

If present, the location of the Trac instance referred to by <tracker>. This option can then be used together with `bzr commit --fixes` to mark bugs in that tracker as being fixed by that commit. For example:

```
trac_twisted_url = http://www.twistedmatrix.com/trac
```

would allow `bzr commit --fixes twisted:1234` to mark Twisted's bug 1234 as fixed.

3.2.4 bugtracker_<tracker>_url

If present, the location of a generic bug tracker instance referred to by <tracker>. The location must contain an `{id}` placeholder, which will be replaced by a specific bug ID. This option can then be used together with `bzr commit --fixes` to mark bugs in that tracker as being fixed by that commit. For example:

```
bugtracker_python_url = http://bugs.python.org/issue{id}
```

would allow `bzr commit --fixes python:1234` to mark bug 1234 in Python's Roundup bug tracker as fixed, or:

```
bugtracker_cpan_url = http://rt.cpan.org/Public/Bug/Display.html?id={id}
```

would allow `bzr commit --fixes cpan:1234` to mark bug 1234 in CPAN's RT bug tracker as fixed, or:

```
bugtracker_hudson_url = http://issues.hudson-ci.org/browse/{id}
```

would allow `bzr commit --fixes hudson:HUDSON-1234` to mark bug HUDSON-1234 in Hudson's JIRA bug tracker as fixed.

3.3 Configuration Settings

3.3.1 Environment settings

While most configuration is handled by configuration files, some options which may be semi-permanent can also be controlled through the environment.

BZR_EMAIL

Override the email id used by Bazaar. Typical format:

```
"John Doe <jdoe@example.com>"
```

See also the `email` configuration option.

BZR_PROGRESS_BAR

Override the progress display. Possible values are “none” or “text”. If the value is “none” then no progress bar is displayed. The value “text” draws the ordinary command line progress bar.

BZR_SIGQUIT_PDB

Control whether SIGQUIT behaves normally or invokes a breakin debugger.

- 0 = Standard SIGQUIT behavior (normally, exit with a core dump)
- 1 = Invoke breakin debugger (default)

BZR_HOME

Override the home directory used by Bazaar.

BZR_SSH

Select a different SSH implementation.

BZR_PDB

Control whether to launch a debugger on error.

- 0 = Standard behavior
- 1 = Launch debugger

BZR_REMOTE_PATH

Path to the Bazaar executable to use when using the bzr+ssh protocol.

See also the `bzr_remote_path` configuration option.

BZR_EDITOR

Path to the editor Bazaar should use for commit messages, etc.

BZR_LOG

Location of the Bazaar log file. You can check the current location by running `bzr version`.

The log file contains debug information that is useful for diagnosing or reporting problems with Bazaar.

Setting this to `NUL` on Windows or `/dev/null` on other platforms will disable logging.

BZR_PLUGIN_PATH

The path to the plugins directory that Bazaar should use. If not set, Bazaar will search for plugins in:

- the user specific plugin directory (containing the `user` plugins),
- the bzrlib directory (containing the `core` plugins),
- the site specific plugin directory if applicable (containing the `site` plugins).

If `BZR_PLUGIN_PATH` is set in any fashion, it will change the the way the plugin are searched.

As for the `PATH` variables, if multiple directories are specified in `BZR_PLUGIN_PATH` they should be separated by the platform specific appropriate character (':' on Unix, ';' on windows)

By default if `BZR_PLUGIN_PATH` is set, it replaces searching in `user`. However it will continue to search in `core` and `site` unless they are explicitly removed.

If you need to change the order or remove one of these directories, you should use special values:

- `-user`, `-core`, `-site` will remove the corresponding path from the default values,
- `+user`, `+core`, `+site` will add the corresponding path before the remaining default values (and also remove it from the default values).

Note that the special values 'user', 'core' and 'site' should be used literally, they will be substituted by the corresponding, platform specific, values.

The examples below use ':' as the separator, windows users should use ';'.

Overriding the default user plugin directory:

```
BZR_PLUGIN_PATH='/path/to/my/other/plugins'
```

Disabling the site directory while retaining the user directory:

```
BZR_PLUGIN_PATH='-site:+user'
```

Disabling all plugins (better achieved with –no-plugins):

```
BZR_PLUGIN_PATH='-user:-core:-site'
```

Overriding the default site plugin directory:

```
BZR_PLUGIN_PATH='/path/to/my/site/plugins:-site':+user
```

BZR_DISABLE_PLUGINS

Under special circumstances (mostly when trying to diagnose a bug), it's better to disable a plugin (or several) rather than uninstalling them completely. Such plugins can be specified in the `BZR_DISABLE_PLUGINS` environment variable.

In that case, `bzr` will stop loading the specified plugins and will raise an import error if they are explicitly imported (by another plugin that depends on them for example).

Disabling `myplugin` and `yourplugin` is achieved by:

```
BZR_DISABLE_PLUGINS='myplugin:yourplugin'
```

BZR_PLUGINS_AT

When adding a new feature or working on a bug in a plugin, developers often need to use a specific version of a given plugin. Since python requires that the directory containing the code is named like the plugin itself this make it impossible to use arbitrary directory names (using a two-level directory scheme is inconvenient). `BZR_PLUGINS_AT` allows such directories even if they don't appear in `BZR_PLUGIN_PATH` .

Plugins specified in this environment variable takes precedence over the ones in `BZR_PLUGIN_PATH`.

The variable specified a list of `plugin_name@plugin path`, `plugin_name` being the name of the plugin as it appears in python module paths, `plugin_path` being the path to the directory containing the plugin code itself (i.e. `plugins/myplugin` not `plugins`). Use ':' as the list separator, use ';' on windows.

Example:

Using a specific version of `myplugin`: `BZR_PLUGINS_AT='myplugin@/home/me/bugfixes/123456-myplugin`

BZRPATH

The path where Bazaar should look for shell plugin external commands.

http_proxy, https_proxy

Specifies the network proxy for outgoing connections, for example:

```
http_proxy=http://proxy.example.com:3128/
https_proxy=http://proxy.example.com:3128/
```

3.3.2 Configuration files

Location

Configuration files are located in `$HOME/.bazaar` on Unix and `C:\Documents and Settings\<username>\Application Data\Bazaar\2.0` on Windows. (You can check the location for your system by using `bzr version`.)

There are three primary configuration files in this location:

- `bazaar.conf` describes default configuration options,
- `locations.conf` describes configuration information for specific branch locations,

- `authentication.conf` describes credential information for remote servers.

Each branch can also contain a configuration file that sets values specific to that branch. This file is found at `.bzr/branch/branch.conf` within the branch. This file is visible to all users of a branch, if you wish to override one of the values for a branch with a setting that is specific to you then you can do so in `locations.conf`.

General format

An ini file has three types of contructs: section headers, section options and comments.

Comments

A comment is any line that starts with a "#" (sometimes called a "hash mark", "pound sign" or "number sign"). Comment lines are ignored by Bazaar when parsing ini files.

Section headers

A section header is a word enclosed in brackets that starts at the begining of a line. A typical section header looks like this:

```
[DEFAULT]
```

The only valid section headers for bazaar.conf currently are [DEFAULT] and [ALIASES]. Section headers are case sensitive. The default section provides for setting options which can be overridden with the branch config file.

For `locations.conf`, the options from the section with the longest matching section header are used to the exclusion of other potentially valid section headers. A section header uses the path for the branch as the section header. Some examples include:

```
[http://mybranches.isp.com/~jdoe/branchdir]
[/home/jdoe/branches/]
```

Section options

A section option resides within a section. A section option contains an option name, an equals sign and a value. For example:

```
email            = John Doe <jdoe@isp.com>
gpg_signing_key  = Amy Pond <amy@example.com>
```

A option can reference other options by enclosing them in curly brackets:

```
my_branch_name = feature_x
my_server      = bzr+ssh://example.com
push_location   = {my_server}/project/{my_branch_name}
```

Option policies

Options defined in a section affect the named directory or URL plus any locations they contain. Policies can be used to change how an option value is interpreted for contained locations. Currently there are three policies available:

none: the value is interpreted the same for contained locations. This is the default behaviour.

norecurse: the value is only used for the exact location specified by the section name.

appendpath: for contained locations, any additional path components are appended to the value.

Policies are specified by keys with names of the form "<option_name>:policy". For example, to define the push location for a tree of branches, the following could be used:

```
[/top/location]
push_location = sftp://example.com/location
push_location:policy = appendpath
```

With this configuration, the push location for `/top/location/branch1` would be `sftp://example.com/location/branch1`.

Section local options

Some options are defined automatically inside a given section and can be refered to in this section only.

For example, the `appendpath` policy can be used like this:

```
[/home/vila/src/bzr/bugs]
mypush = lp:~vila/bzr
mypush:policy=appendpath
```

Using `relpath` to achieve the same result is done like this:

```
[/home/vila/src/bzr/bugs]
mypush = lp:~vila/bzr/{relpath}
```

In both cases, when used in a directory like `/home/vila/src/bzr/bugs/832013-expand-in-stack` we'll get:

```
$ bzr config mypush
lp:~vila/bzr/832013-expand-in-stack
```

Another such option is `basename` which can be used like this:

```
[/home/vila/src/bzr]
mypush = lp:~vila/bzr/{basename}
```

When used in a directory like `/home/vila/src/bzr/bugs/832013-expand-in-stack` we'll get:

```
$ bzr config mypush
lp:~vila/bzr/832013-expand-in-stack
```

Note that `basename` here refers to the base name of `relpath` which itself is defined as the relative path between the section name and the location it matches.

Another such option is `branchname`, which refers to the name of a colocated branch. For non-colocated branches, it behaves like basename. It can be used like this:

```
[/home/vila/src/bzr/bugs]
mypush = lp:~vila/bzr/{branchname}
```

When used with a colocated branch named `832013-expand-in-stack`, we'll get:

```
bzr config mypush
lp:~vila/bzr/832013-expand-in-stack
```

When an option is local to a Section, it cannot be referred to from option values in any other section from the same `Store` nor from any other `Store`.

The main configuration file, bazaar.conf

`bazaar.conf` allows two sections: `[DEFAULT]` and `[ALIASES]`. The default section contains the default configuration options for all branches. The default section can be overriden by providing a branch-specific section in `locations.conf`.

A typical `bazaar.conf` section often looks like the following:

```
[DEFAULT]
email             = John Doe <jdoe@isp.com>
editor            = /usr/bin/vim
create_signatures = when-required
```

The branch location configuration file, locations.conf

`locations.conf` allows one to specify overriding settings for a specific branch. The format is almost identical to the default section in bazaar.conf with one significant change: The section header, instead of saying default, will be the path to a branch that you wish to override a value for. The '?' and '*' wildcards are supported:

```
[/home/jdoe/branches/nethack]
email = Nethack Admin <nethack@nethack.com>

[http://hypothetical.site.com/branches/devel-branch]
create_signatures = always
```

The authentication configuration file, authentication.conf

`authentication.conf` allows one to specify credentials for remote servers. This can be used for all the supported transports and any part of bzr that requires authentication (smtp for example).

The syntax of the file obeys the same rules as the others except for the option policies which don't apply.

For more information on the possible uses of the authentication configuration file see *Authentication Settings*.

3.3.3 Common options

debug_flags

A comma-separated list of debugging options to turn on. The same values can be used as with the -D command-line option (see *help global-options*). For example:

```
debug_flags = hpss
```

or:

```
debug_flags = hpss,evil
```

email

The email address to use when committing a branch. Typically takes the form of:

```
email = Full Name <account@hostname.tld>
```

editor

The path of the editor that you wish to use if *bzr commit* is run without a commit message. This setting is trumped by the environment variable `BZR_EDITOR`, and overrides the `VISUAL` and `EDITOR` environment variables.

log_format

The default log format to use. Standard log formats are `long`, `short` and `line`. Additional formats may be provided by plugins. The default value is `long`.

check_signatures

Reserved for future use. These options will allow a policy for branches to require signatures.

require The gnupg signature for revisions must be present and must be valid.

ignore Do not check gnupg signatures of revisions.

check-available (default) If gnupg signatures for revisions are present, check them. Bazaar will fail if it finds a bad signature, but will not fail if no signature is present.

create_signatures

Defines the behaviour of signing revisions on commits. By default bzr will not sign new commits.

always Sign every new revision that is committed. If the signing fails then the commit will not be made.

when-required Reserved for future use.

never Reserved for future use.

In future it is planned that `when-required` will sign newly committed revisions only when the branch requires them. `never` will refuse to sign newly committed revisions, even if the branch requires signatures.

dirstate.fdatasync

If true (default), working tree metadata changes are flushed through the OS buffers to physical disk. This is somewhat slower, but means data should not be lost if the machine crashes. See also repository.fdatasync.

gpg_signing_key

The GnuPG user identity to use when signing commits. Can be an e-mail address, key fingerprint or full key ID. When unset or when set to "default" Bazaar will use the user e-mail set with `whoami`.

recurse

Only useful in `locations.conf`. Defines whether or not the configuration for this section applies to subdirectories:

true (default) This section applies to subdirectories as well.

false This section only applies to the branch at this directory and not branches below it.

gpg_signing_command

(Default: "gpg"). Which program should be used to sign and check revisions. For example:

```
gpg_signing_command = /usr/bin/gnpg
```

The specified command must accept the options "–clearsign" and "-u <email>".

bzr_remote_path

(Default: "bzr"). The path to the command that should be used to run the smart server for bzr. This value may only be specified in locations.conf, because:

- it's needed before branch.conf is accessible
- allowing remote branch.conf files to specify commands would be a security risk

It is overridden by the BZR_REMOTE_PATH environment variable.

smtp_server

(Default: "localhost"). SMTP server to use when Bazaar needs to send email, eg. with `merge-directive --mail-to`, or the bzr-email plugin.

smtp_username, smtp_password

User and password to authenticate to the SMTP server. If smtp_username is set, and smtp_password is not, Bazaar will prompt for a password. These settings are only needed if the SMTP server requires authentication to send mail.

locks.steal_dead

If set to true, bzr will automatically break locks held by processes from the same machine and user that are no longer alive. Otherwise, it will print a message and you can break the lock manually, if you are satisfied the object is no longer in use.

mail_client

A mail client to use for sending merge requests. By default, bzr will try to use `mapi` on Windows. On other platforms, it will try `xdg-email`. If either of these fails, it will fall back to `editor`.

Supported values for specific clients:

claws Use Claws. This skips a dialog for attaching files.

evolution Use Evolution.

kmail Use KMail.

mutt Use Mutt.

thunderbird Use Mozilla Thunderbird or Icedove. For Thunderbird/Icedove 1.5, this works around some bugs that xdg-email doesn't handle.

Supported generic values are:

default See above.

editor Use your editor to compose the merge request. This also uses your commit id, (see `bzr whoami`), smtp_server and (optionally) smtp_username and smtp_password.

mapi Use your preferred e-mail client on Windows.

xdg-email Use xdg-email to run your preferred mail program

repository.fdatasync

If true (default), repository changes are flushed through the OS buffers to physical disk. This is somewhat slower, but means data should not be lost if the machine crashes. See also dirstate.fdatasync.

submit_branch

The branch you intend to submit your current work to. This is automatically set by `bzr send`, and is also used by the `submit:` revision spec. This should usually be set on a per-branch or per-location basis.

public_branch

A publically-accessible version of this branch (implying that this version is not publically-accessible). Used (and set) by `bzr send`.

suppress_warnings

A list of strings, each string represent a warning that can be emitted by bzr. Mentioning a warning in this list tells bzr to not emit it.

Valid values:

- **`format_deprecation`:** whether the format deprecation warning is shown on repositories that are using deprecated formats.

default_format

A format name for the default format used when creating branches. See `bzr help formats` for possible values.

3.3.4 Unicode options

output_encoding

A Python unicode encoding name for text output from bzr, such as log information. Values include: utf8, cp850, ascii, iso-8859-1. The default is the terminal encoding prefered by the operating system.

3.3.5 Branch type specific options

These options apply only to branches that use the `dirstate-tags` or later format. They are usually set in `.bzr/branch/branch.conf` automatically, but may be manually set in `locations.conf` or `bazaar.conf`.

append_revisions_only

If set to "True" then revisions can only be appended to the log, not removed. A branch with this setting enabled can only pull from another branch if the other branch's log is a longer version of its own. This is normally set by `bzr init --append-revisions-only`. If you set it manually, use either 'True' or 'False' (case-sensitive) to maintain compatibility with previous bzr versions (older than 2.2).

parent_location

If present, the location of the default branch for pull or merge. This option is normally set when creating a branch, the first `pull` or by `pull --remember`.

push_location

If present, the location of the default branch for push. This option is normally set by the first `push` or `push --remember`.

push_strict

If present, defines the `--strict` option default value for checking uncommitted changes before pushing.

dpush_strict

If present, defines the `--strict` option default value for checking uncommitted changes before pushing into a different VCS without any custom bzr metadata.

bound_location

The location that commits should go to when acting as a checkout. This option is normally set by `bind`.

bound

If set to "True", the branch should act as a checkout, and push each commit to the bound_location. This option is normally set by `bind`/`unbind`.

send_strict

If present, defines the `--strict` option default value for checking uncommitted changes before sending a merge directive.

add.maximum_file_size

Defines the maximum file size the command line "add" operation will allow in recursive mode, with files larger than this value being skipped. You may specify this value as an integer (in which case it is interpreted as bytes), or you may specify the value using SI units, i.e. 10KB, 20MB, 1G. A value of 0 will disable skipping.

3.3.6 External Merge Tools

bzr.mergetool.<name>

Defines an external merge tool called <name> with the given command-line. Arguments containing spaces should be quoted using single or double quotes. The executable may omit its path if it can be found on the PATH.

The following markers can be used in the command-line to substitute filenames involved in the merge conflict:

```
{base}      file.BASE
{this}      file.THIS
{other}     file.OTHER
{result}    output file
{this_temp} temp copy of file.THIS, used to overwrite output file if merge
            succeeds.
```

For example:

```
bzr.mergetool.kdiff3 = kdiff3 {base} {this} {other} -o {result}
```

Because `mergetool` and `config` itself both use curly braces as interpolation markers, trying to display the mergetool line results in the following problem:

```
$ bzr config bzr.mergetool.kdiff3='kdiff3 {base} {this} {other} -o {result}'
$ bzr config bzr.mergetool.kdiff3
bzr: ERROR: Option base is not defined while expanding "kdiff3 {base} {this} {other} -o {result}".
```

To avoid this, `config` can be instructed not to try expanding variables:

```
$ bzr config --all bzr.mergetool.kdiff3
branch:
  bzr.mergetool.kdiff3 = kdiff3 {base} {this} {other} -o {result}
```

bzr.default_mergetool

Specifies which external merge tool (as defined above) should be selected by default in tools such as `bzr qconflicts`.

For example:

```
bzr.default_mergetool = kdiff3
```

3.4 Conflict Types

Some operations, like merge, revert and pull, modify the contents of your working tree. These modifications are programmatically generated, and so they may conflict with the current state of your working tree.

When conflicts are present in your working tree (as shown by `bzr conflicts`), you should resolve them and then inform bzr that the conflicts have been resolved.

Resolving conflicts is sometimes not obvious. Either because the user that should resolve them is not the one responsible for their occurrence, as is the case when merging other people's work or because some conflicts are presented in a way that is not easy to understand.

Bazaar tries to avoid conflicts ; its aim is to ask you to resolve the conflict if and only if there's an actual conceptual conflict in the source tree. Because Bazaar doesn't understand the real meaning of the files being versioned, it can,

when faced with ambiguities, fall short in either direction trying to resolve the conflict itself. Many kinds of changes can be combined programmatically, but sometimes only a human can determine the right thing to do.

When Bazaar generates a conflict, it adds information into the working tree to present the conflicting versions, and it's up to you to find the correct resolution.

Whatever the conflict is, resolving it is roughly done in two steps:

1. Modify the working tree content so that the conflicted item is now in the state you want to keep, then
2. Inform Bazaar that the conflict is now solved and ask to cleanup any remaining generated information (`bzr resolve <item>`).

For most conflict types, there are some obvious ways to modify the working tree and put it into the desired state. For some types of conflicts, Bazaar itself already made a choice, when possible.

Yet, whether Bazaar makes a choice or not, there are some other simple but different ways to resolve the conflict.

Each type of conflict is explained below, and the action which must be done to resolve the conflict is outlined.

Various actions are available depending on the kind of conflict, for some of these actions, Bazaar can provide some help. In the end you should at least inform Bazaar that you're done with the conflict with:

```
``bzr resolve FILE --action=done'
```

Note that this is the default action when a single file is involved so you can simply use:

```
``bzr resolve FILE``
```

See `bzr help resolve` for more details.

3.4.1 Text conflicts

Typical message:

```
Text conflict in FILE
```

These are produced when a text merge cannot completely reconcile two sets of text changes. Bazaar will emit files for each version with the extensions THIS, OTHER, and BASE. THIS is the version of the file from the target tree, i.e. the tree that you are merging changes into. OTHER is the version that you are merging into the target. BASE is an older version that is used as a basis for comparison.

In the main copy of the file, Bazaar will include all the changes that it could reconcile, and any un-reconciled conflicts are surrounded by "herringbone" markers like <<<<<<<.

Say the initial text is "The project leader released it.", and THIS modifies it to "Martin Pool released it.", while OTHER modifies it to "The project leader released Bazaar." A conflict would look like this:

```
<<<<<<< TREE
Martin Pool released it.
=======
The project leader released Bazaar.
>>>>>>> MERGE-SOURCE
```

The correct resolution would be "Martin Pool released Bazaar."

You can handle text conflicts either by editing the main copy of the file, or by invoking external tools on the THIS, OTHER and BASE versions. It's worth mentioning that resolving text conflicts rarely involves picking one set of changes over the other (but see below when you encounter these cases). More often, the two sets of changes must be intelligently combined.

If you edit the main copy, be sure to remove the herringbone markers. When you are done editing, the file should look like it never had a conflict, and be ready to commit.

When you have resolved text conflicts, just run `bzr resolve --auto`, and Bazaar will auto-detect which conflicts you have resolved.

When the conflict is resolved, Bazaar deletes the previously generated `.BASE`, `.THIS` and `.OTHER` files if they are still present in the working tree.

When you want to pick one set of changes over the other, you can use `bzr resolve` with one of the following actions:

- `--action=take-this` will issue `mv FILE.THIS FILE`,
- `--action=take-other` will issue `mv FILE.OTHER FILE`.

Note that if you have modified `FILE.THIS` or `FILE.OTHER`, these modifications will be taken into account.

3.4.2 Content conflicts

Typical message:

```
Contents conflict in FILE
```

This conflict happens when there are conflicting changes in the working tree and the merge source, but the conflicted items are not text files. They may be binary files, or symlinks, or directories. It can even happen with files that are deleted on one side, and modified on the other.

Like text conflicts, Bazaar will emit THIS, OTHER and BASE files. (They may be regular files, symlinks or directories). But it will not include a "main copy" of the file with herringbone conflict markers. It will appear that the "main copy" has been renamed to THIS or OTHER.

To resolve that kind of conflict, you should rebuild FILE from either version or a combination of both.

`bzr resolve` recognizes the following actions:

- `--action=take-this` will issue `bzr mv FILE.THIS FILE`,
- `--action=take-other` will issue `bzr mv FILE.OTHER FILE`,
- `--action=done` will just mark the conflict as resolved.

Any action will also delete the previously generated `.BASE`, `.THIS` and `.OTHER` files if they are still present in the working tree.

Bazaar cannot auto-detect when conflicts of this kind have been resolved.

3.4.3 Tag conflicts

Typical message:

```
Conflicting tags:
    version-0.1
```

When pulling from or pushing to another branch, Bazaar informs you about tags that conflict between the two branches; that is the same tag points to two different revisions. You need not resolve these conflicts, but subsequent uses of pull or push will result in the same message.

To resolve the conflict, you must apply the correct tags to either the target branch or the source branch as appropriate. Use "bzr tags –show-ids -d SOURCE_URL" to see the tags in the source branch. If you want to make the target branch's tags match the source branch, then in the target branch do `bzr tag --force`

`-r revid:REVISION_ID CONFLICTING_TAG` for each of the CONFLICTING_TAGs, where REVISION_ID comes from the list of tags in the source branch. You need not call "bzr resolve" after doing this. To resolve in favor of the target branch, you need to similarly use `tag --force` in the source branch. (Note that pulling or pushing using –overwrite will overwrite all tags as well.)

3.4.4 Duplicate paths

Typical message:

```
Conflict adding file FILE.  Moved existing file to FILE.moved.
```

Sometimes Bazaar will attempt to create a file using a pathname that has already been used. The existing file will be renamed to "FILE.moved".

To resolve that kind of conflict, you should rebuild FILE from either version or a combination of both.

`bzr resolve` recognizes the following actions:

- `--action=take-this` will issue `bzr rm FILE ; bzr mv FILE.moved FILE`,
- `--action=take-other` will issue `bzr rm FILE.moved`,
- `--action=done` will just mark the conflict as resolved.

Note that you must get rid of FILE.moved before using `--action=done`.

Bazaar cannot auto-detect when conflicts of this kind have been resolved.

3.4.5 Unversioned parent

Typical message:

```
Conflict because FILE is not versioned, but has versioned children.
```

Sometimes Bazaar will attempt to create a file whose parent directory is not versioned. This happens when the directory has been deleted in the target, but has a new child in the source, or vice versa. In this situation, Bazaar will version the parent directory as well. Resolving this issue depends very much on the particular scenario. You may wish to rename or delete either the file or the directory. When you are satisfied, you can run "bzr resolve FILE" to mark the conflict as resolved.

3.4.6 Missing parent

Typical message:

```
Conflict adding files to FILE.  Created directory.
```

This happens when a directory has been deleted in the target, but has new children in the source. This is similar to the "unversioned parent" conflict, except that the parent directory does not *exist*, instead of just being unversioned. In this situation, Bazaar will create the missing parent. Resolving this issue depends very much on the particular scenario.

To resolve that kind of conflict, you should either remove or rename the children or the directory or a combination of both.

`bzr resolve` recognizes the following actions:

- `--action=take-this` will issue `bzr rm directory` including the children,
- `--action=take-other` will acknowledge Bazaar choice to keep the children and restoring the directory,

- `--action=done` will just mark the conflict as resolved.

Bazaar cannot auto-detect when conflicts of this kind have been resolved.

3.4.7 Deleting parent

Typical message:

```
Conflict: can't delete DIR because it is not empty.  Not deleting.
```

This is the opposite of "missing parent". A directory is deleted in the source, but has new children in the target (either because a directory deletion is merged or because the merge introduce new children). Bazaar will retain the directory. Resolving this issue depends very much on the particular scenario.

To resolve that kind of conflict, you should either remove or rename the children or the directory or a combination of both.

`bzr resolve` recognizes the following actions:

- `--action=take-this` will acknowledge Bazaar choice to keep the directory,
- `--action=take-other` will issue `bzr rm directory` including the children,
- `--action=done` will just mark the conflict as resolved.

Note that when merging a directory deletion, if unversioned files are present, they become potential orphans has they don't have a directory parent anymore.

Handling such orphans, *before* the conflict is created, is controlled by setting the `bzr.transform.orphan_policy` configuration option.

There are two possible values for this option:

- `conflict` (the default): will leave the orphans in place and generate a conflicts,
- `move`: will move the orphans to a `bzr-orphans` directory at the root of the working tree with names like `<file>.~#~`.

Bazaar cannot auto-detect when conflicts of this kind have been resolved.

3.4.8 Path conflict

Typical message:

```
Path conflict: PATH1 / PATH2
```

This happens when the source and target have each modified the name or parent directory of a file. Bazaar will use the path elements from the source.

To resolve that kind of conflict, you just have to decide what name should be retained for the file involved.

`bzr resolve` recognizes the following actions:

- `--action=take-this` will revert Bazaar choice and keep `PATH1` by issuing `bzr mv PATH2 PATH1`,
- `--action=take-other` will acknowledge Bazaar choice of keeping `PATH2`,
- `--action=done` will just mark the conflict as resolved.

Bazaar cannot auto-detect when conflicts of this kind have been resolved.

3.4.9 Parent loop

Typical message:

```
Conflict moving FILE into DIRECTORY.  Cancelled move.
```

This happens when the source and the target have each moved directories, so that, if the change could be applied, a directory would be contained by itself. For example:

```
$ bzr init
$ bzr mkdir white
$ bzr mkdir black
$ bzr commit -m "BASE"
$ bzr branch . ../other
$ bzr mv white black
$ bzr commit -m "THIS"
$ bzr mv ../other/black ../other/white
$ bzr commit ../other -m "OTHER"
$ bzr merge ../other
```

In this situation, Bazaar will cancel the move, and leave `white` in `black`. To resolve that kind of conflict, you just have to decide what name should be retained for the directories involved.

`bzr resolve` recognizes the following actions:

- `--action=take-this` will acknowledge Bazaar choice of leaving `white` in `black`,
- **`--action=take-other` will revert Bazaar choice and move `black` in** `white` by issuing `bzr mv black/white white ; bzr mv black white`,
- `--action=done` will just mark the conflict as resolved.

Bazaar cannot auto-detect when conflicts of this kind have been resolved.

3.4.10 Non-directory parent

Typical message:

```
Conflict: foo.new is not a directory, but has files in it.
Created directory.
```

This happens when one side has added files to a directory, and the other side has changed the directory into a file or symlink. For example:

```
$ bzr init
$ bzr mkdir foo
$ bzr commit -m "BASE"
$ bzr branch . ../other
$ rmdir foo
$ touch foo
$ bzr commit -m "THIS"
$ bzr mkdir ../other/foo/bar
$ bzr commit ../other -m "OTHER"
$ bzr merge ../other
```

To resolve that kind of conflict, you have to decide what name should be retained for the file, directory or symlink involved.

`bzr resolve` recognizes the following actions:

- `--action=take-this` will issue `bzr rm --force foo.new` and `bzr add foo`,

- `--action=take-other` will issue `bzr rm --force foo` and `bzr mv foo.new foo`,
- `--action=done` will just mark the conflict as resolved.

Bazaar cannot auto-detect when conflicts of this kind have been resolved.

3.4.11 MalformedTransform

It is possible (though very rare) for Bazaar to raise a MalformedTransform exception. This means that Bazaar encountered a filesystem conflict that it was unable to resolve. This usually indicates a bug. Please let us know if you encounter this. Our bug tracker is at https://launchpad.net/bzr/+bugs

3.5 Current Storage Formats

2a (native) (default) Format for the bzr 2.0 series. Uses group- compress storage. Provides rich roots which are a one-way transition.

See *Storage Formats* for more about storage formats.

3.6 Debug Flags

These flags can be passed on the bzr command line or (without the `-D` prefix) put in the `debug_flags` variable in `bazaar.conf`.

-Dauth	Trace authentication sections used.
-Dbytes	Print out how many bytes were transferred
-Ddirstate	Trace dirstate activity (verbose!)
-Derror	Instead of normal error handling, always print a traceback on error.
-Devil	Capture call sites that do expensive or badly-scaling operations.
-Dfetch	Trace history copying between repositories.
-Dfilters	Emit information for debugging content filtering.
-Dforceinvdeltas	Force use of inventory deltas during generic streaming fetch.
-Dgraph	Trace graph traversal.
-Dhashcache	Log every time a working file is read to determine its hash.
-Dhooks	Trace hook execution.
-Dhpss	Trace smart protocol requests and responses.
-Dhpssdetail	More hpss details.
-Dhpssvfs	Traceback on vfs access to Remote objects.
-Dhttp	Trace http connections, requests and responses.
-Dindex	Trace major index operations.
-Dknit	Trace knit operations.
-Dlock	Trace when lockdir locks are taken or released.
-Dnoretry	If a connection is reset, fail immediately rather than retrying the request.

-Dprogress	Trace progress bar operations.
-Dmem_dump	Dump memory to a file upon an out of memory error.
-Dmerge	Emit information for debugging merges.
-Dno_apport	Don't use apport to report crashes.
-Dno_activity	Don't show transport activity indicator in progress bar.
-Dpack	Emit information about pack operations.
-Drelock	Emit a message every time a branch or repository object is unlocked then relocked the same way.
-Dsftp	Trace SFTP internals.
-Dstatic_tuple	Error when a tuple is used where a StaticTuple is expected
-Dstream	Trace fetch streams.
-Dstrict_locks	Trace when OS locks are potentially used in a non-portable manner.
-Dunlock	Some errors during unlock are treated as warnings.
-DIDS_never	Never use InterDifferingSerializer when fetching.
-DIDS_always	Always use InterDifferingSerializer to fetch if appropriate for the format, even for non-local fetches.

3.7 Environment Variables

See bzr help configuration for more details.

BZRPATH	Path where bzr is to look for shell plugin external commands.
BZR_EMAIL	E-Mail address of the user. Overrides EMAIL.
EMAIL	E-Mail address of the user.
BZR_EDITOR	Editor for editing commit messages. Overrides EDITOR.
EDITOR	Editor for editing commit messages.
BZR_PLUGIN_PATH	Paths where bzr should look for plugins.
BZR_DISABLE_PLUGINS	Plugins that bzr should not load.
BZR_PLUGINS_AT	Plugins to load from a directory not in BZR_PLUGIN_PATH.
BZR_HOME	Directory holding .bazaar config dir. Overrides HOME.
BZR_HOME (Win32)	Directory holding bazaar config dir. Overrides APPDATA and HOME.
BZR_REMOTE_PATH	Full name of remote 'bzr' command (for bzr+ssh:// URLs).
BZR_SSH	Path to SSH client, or one of paramiko, openssh, sshcorp, plink or lsh.
BZR_LOG	Location of .bzr.log (use '/dev/null' to suppress log).
BZR_LOG (Win32)	Location of .bzr.log (use 'NUL' to suppress log).
BZR_COLUMNS	Override implicit terminal width.
BZR_CONCURRENCY	Number of processes that can be run concurrently (selftest)
BZR_PROGRESS_BAR	Override the progress display. Values are 'none' or 'text'.
BZR_PDB	Control whether to launch a debugger on error.
BZR_SIGQUIT_PDB	Control whether SIGQUIT behaves normally or invokes a breakin debugger.
BZR_TEXTUI_INPUT	Force console input mode for prompts to line-based (instead of char-based).

3.8 Files

On Unix ~/.bazaar/bazaar.conf

On Windows C:\Documents and Settings\username\Application Data\bazaar\2.0\bazaar.conf

Contains the user's default configuration. The section `[DEFAULT]` is used to define general configuration that will be applied everywhere. The section `[ALIASES]` can be used to create command aliases for commonly used options.

A typical config file might look something like:

```
[DEFAULT]
email=John Doe <jdoe@isp.com>

[ALIASES]
commit = commit --strict
log10 = log --short -r -10..-1
```

3.9 Global Options

These options may be used with any command, and may appear in front of any command. (e.g. `bzr --profile help`).

--version Print the version number. Must be supplied before the command.

--no-aliases Do not process command aliases when running this command.

--builtin Use the built-in version of a command, not the plugin version. This does not suppress other plugin effects.

--no-plugins Do not process any plugins.

--no-l10n Do not translate messages.

--concurrency Number of processes that can be run concurrently (selftest).

--profile Profile execution using the hotshot profiler.

--lsprof Profile execution using the lsprof profiler.

--lsprof-file Profile execution using the lsprof profiler, and write the results to a specified file. If the filename ends with ".txt", text format will be used. If the filename either starts with "callgrind.out" or end with ".callgrind", the output will be formatted for use with KCacheGrind. Otherwise, the output will be a pickle.

--coverage Generate line coverage report in the specified directory.

-Oname=value Override the `name` config option setting it to `value` for the duration of the command. This can be used multiple times if several options need to be overridden.

See http://doc.bazaar.canonical.com/developers/profiling.html for more information on profiling.

A number of debug flags are also available to assist troubleshooting and development. See *Debug Flags*.

3.10 Hooks

3.10.1 Introduction

A hook of type *xxx* of class *yyy* needs to be registered using:

```
yyy.hooks.install_named_hook("xxx", ...)
```

See `Using hooks` in the User Guide for examples.

The class that contains each hook is given before the hooks it supplies. For instance, BranchHooks as the class is the hooks class for *bzrlib.branch.Branch.hooks*.

Each description also indicates whether the hook runs on the client (the machine where bzr was invoked) or the server (the machine addressed by the branch URL). These may be, but are not necessarily, the same machine.

Plugins (including hooks) are run on the server if all of these is true:

- The connection is via a smart server (accessed with a URL starting with "bzr://", "bzr+ssh://" or "bzr+http://", or accessed via a "http://" URL when a smart server is available via HTTP).
- The hook is either server specific or part of general infrastructure rather than client specific code (such as commit).

3.10.2 BranchHooks

automatic_tag_name

Introduced in: 2.2

Called to determine an automatic tag name for a revision. automatic_tag_name is called with (branch, revision_id) and should return a tag name or None if no tag name could be determined. The first non-None tag name returned will be used.

open

Introduced in: 1.8

Called with the Branch object that has been opened after a branch is opened.

post_branch_init

Introduced in: 2.2

Called after new branch initialization completes. post_branch_init is called with a bzrlib.branch.BranchInitHookParams. Note that init, branch and checkout (both heavyweight and lightweight) will all trigger this hook.

post_change_branch_tip

Introduced in: 1.4

Called in bzr client and server after a change to the tip of a branch is made. post_change_branch_tip is called with a bzrlib.branch.ChangeBranchTipParams. Note that push, pull, commit, uncommit will all trigger this hook.

post_commit

Introduced in: 0.15

Called in the bzr client after a commit has completed. post_commit is called with (local, master, old_revno, old_revid, new_revno, new_revid). old_revid is NULL_REVISION for the first commit to a branch.

post_pull

Introduced in: 0.15

Called after a pull operation completes. post_pull is called with a bzrlib.branch.PullResult object and only runs in the bzr client.

post_push

Introduced in: 0.15

Called after a push operation completes. post_push is called with a bzrlib.branch.BranchPushResult object and only runs in the bzr client.

post_switch

Introduced in: 2.2

Called after a checkout switches branch. post_switch is called with a bzrlib.branch.SwitchHookParams.

post_uncommit

Introduced in: 0.15

Called in the bzr client after an uncommit completes. post_uncommit is called with (local, master, old_revno, old_revid, new_revno, new_revid) where local is the local branch or None, master is the target branch, and an empty branch receives new_revno of 0, new_revid of None.

pre_change_branch_tip

Introduced in: 1.6

Called in bzr client and server before a change to the tip of a branch is made. pre_change_branch_tip is called with a bzrlib.branch.ChangeBranchTipParams. Note that push, pull, commit, uncommit will all trigger this hook.

pre_commit

Introduced in: 0.91

Called after a commit is calculated but before it is completed. pre_commit is called with (local, master, old_revno, old_revid, future_revno, future_revid, tree_delta, future_tree). old_revid is NULL_REVISION for the first commit to a branch, tree_delta is a TreeDelta object describing changes from the basis revision. hooks MUST NOT modify this delta. future_tree is an in-memory tree obtained from CommitBuilder.revision_tree() and hooks MUST NOT modify this tree.

transform_fallback_location

Introduced in: 1.9

Called when a stacked branch is activating its fallback locations. transform_fallback_location is called with (branch, url), and should return a new url. Returning the same url allows it to be used as-is, returning a different one can be used to cause the branch to stack on a closer copy of that fallback_location. Note that the branch cannot have history accessing methods called on it during this hook because the fallback locations have not been activated. When there are multiple hooks installed for transform_fallback_location, all are called with the url returned from the previous hook.The order is however undefined.

3.10.3 CommandHooks

extend_command

Introduced in: 1.13

Called after creating a command object to allow modifications such as adding or removing options, docs etc. Called with the new bzrlib.commands.Command object.

get_command

Introduced in: 1.17

Called when creating a single command. Called with (cmd_or_None, command_name). get_command should either return the cmd_or_None parameter, or a replacement Command object that should be used for the command. Note that the Command.hooks hooks are core infrastructure. Many users will prefer to use bzrlib.commands.register_command or plugin_cmds.register_lazy.

get_missing_command

Introduced in: 1.17

Called when creating a single command if no command could be found. Called with (command_name). get_missing_command should either return None, or a Command object to be used for the command.

list_commands

Introduced in: 1.17

Called when enumerating commands. Called with a set of cmd_name strings for all the commands found so far. This set is safe to mutate - e.g. to remove a command. list_commands should return the updated set of command names.

post_command

Introduced in: 2.6

Called after executing a command. Called with the command object.

pre_command

Introduced in: 2.6

Called prior to executing a command. Called with the command object.

3.10.4 _ConfigHooks

get

Introduced in: 2.4

Invoked when a config option is read. The signature is (stack, name, value).

load

Introduced in: 2.4

Invoked when a config store is loaded. The signature is (store).

remove

Introduced in: 2.4

Invoked when a config option is removed. The signature is (stack, name).

save

Introduced in: 2.4

Invoked when a config store is saved. The signature is (store).

set

Introduced in: 2.4

Invoked when a config option is set. The signature is (stack, name, value).

3.10.5 ControlDirHooks

post_repo_init

Introduced in: 2.2

Invoked after a repository has been initialized. post_repo_init is called with a bzrlib.controldir.RepoInitHookParams.

pre_open

Introduced in: 1.14

Invoked before attempting to open a ControlDir with the transport that the open will use.

3.10.6 InfoHooks

repository

Introduced in: 1.15

Invoked when displaying the statistics for a repository. repository is called with a statistics dictionary as returned by the repository and a file-like object to write to.

3.10.7 LockHooks

lock_acquired

Introduced in: 1.8

Called with a bzrlib.lock.LockResult when a physical lock is acquired.

lock_broken

Introduced in: 1.15

Called with a bzrlib.lock.LockResult when a physical lock is broken.

lock_released

Introduced in: 1.8

Called with a bzrlib.lock.LockResult when a physical lock is released.

3.10.8 MergeHooks

merge_file_content

Introduced in: 2.1

Called with a bzrlib.merge.Merger object to create a per file merge object when starting a merge. Should return either None or a subclass of `bzrlib.merge.AbstractPerFileMerger`. Such objects will then be called per file that needs to be merged (including when one side has deleted the file and the other has changed it). See the AbstractPerFileMerger API docs for details on how it is used by merge.

post_merge

Introduced in: 2.5

Called after a merge. Receives a Merger object as the single argument. The return value is ignored.

pre_merge

Introduced in: 2.5

Called before a merge. Receives a Merger object as the single argument.

3.10.9 MergeDirectiveHooks

merge_request_body

Introduced in: 1.15.0

Called with a MergeRequestBodyParams when a body is needed for a merge request. Callbacks must return a body. If more than one callback is registered, the output of one callback is provided to the next.

3.10.10 MessageEditorHooks

commit_message_template

Introduced in: 1.10

Called when a commit message is being generated. commit_message_template is called with the bzrlib.commit.Commit object and the message that is known so far. commit_message_template must return a new message to use (which could be the same as it was given). When there are multiple hooks registered for commit_message_template, they are chained with the result from the first passed into the second, and so on.

set_commit_message

Introduced in: 2.4

Set a fixed commit message. set_commit_message is called with the bzrlib.commit.Commit object (so you can also change e.g. revision properties by editing commit.builder._revprops) and the message so far. set_commit_message must return the message to use or None if it should use the message editor as normal.

3.10.11 MutableTreeHooks

post_build_tree

Introduced in: 2.5

Called after a completely new tree is built. The hook is called with the tree as its only argument.

post_commit

Introduced in: 2.0

Called after a commit is performed on a tree. The hook is called with a bzrlib.mutabletree.PostCommitHookParams object. The mutable tree the commit was performed on is available via the mutable_tree attribute of that object.

post_transform

Introduced in: 2.5

Called after a tree transform has been performed on a tree. The hook is called with the tree that is being transformed and the transform.

pre_transform

Introduced in: 2.5

Called before a tree transform on this tree. The hook is called with the tree that is being transformed and the transform.

start_commit

Introduced in: 1.4

Called before a commit is performed on a tree. The start commit hook is able to change the tree before the commit takes place. start_commit is called with the bzrlib.mutabletree.MutableTree that the commit is being performed on.

3.10.12 SmartClientHooks

call

Introduced in: unknown

Called when the smart client is submitting a request to the smart server. Called with a bzrlib.smart.client.CallHookParams object. Streaming request bodies, and responses, are not accessible.

3.10.13 SmartServerHooks

server_exception

Introduced in: 2.4

Called by the bzr server when an exception occurs. server_exception is called with the sys.exc_info() tuple return true for the hook if the exception has been handled, in which case the server will exit normally.

server_started

Introduced in: 0.16

Called by the bzr server when it starts serving a directory. server_started is called with (backing urls, public url), where backing_url is a list of URLs giving the server-specific directory locations, and public_url is the public URL for the directory being served.

server_started_ex

Introduced in: 1.17

Called by the bzr server when it starts serving a directory. server_started is called with (backing_urls, server_obj).

server_stopped

Introduced in: 0.16

Called by the bzr server when it stops serving a directory. server_stopped is called with the same parameters as the server_started hook: (backing_urls, public_url).

3.10.14 StatusHooks

post_status

Introduced in: 2.3

Called with argument StatusHookParams after Bazaar has displayed the status. StatusHookParams has the attributes (old_tree, new_tree, to_file, versioned, show_ids, short, verbose). The last four arguments correspond to the command line options specified by the user for the status command. to_file is the output stream for writing.

pre_status

Introduced in: 2.3

Called with argument StatusHookParams before Bazaar displays the status. StatusHookParams has the attributes (old_tree, new_tree, to_file, versioned, show_ids, short, verbose). The last four arguments correspond to the command line options specified by the user for the status command. to_file is the output stream for writing.

3.10.15 TransportHooks

post_connect

Introduced in: 2.5

Called after a new connection is established or a reconnect occurs. The sole argument passed is either the connected transport or smart medium instance.

3.10.16 RioVersionInfoBuilderHooks

revision

Introduced in: 1.15

Invoked when adding information about a revision to the RIO stanza that is printed. revision is called with a revision object and a RIO stanza.

3.11 Location aliases

Bazaar defines several aliases for locations associated with a branch. These can be used with most commands that expect a location, such as *bzr push*.

The aliases are:

```
:bound     The branch this branch is bound to, for bound branches.
:parent    The parent of this branch.
:public    The public location of this branch.
:push      The saved location used for `bzr push` with no arguments.
:submit    The submit branch for this branch.
:this      This branch.
```

For example, to push to the parent location:

```
bzr push :parent
```

3.12 Log Formats

A log format controls how information about each revision is displayed. The standard log formats are compared below:

Feature	long	short	line
design goal	detailed view	concise view	1 revision per line
committer	name+email	name only	name only
author	name+email	-	-
date-time format	full	date only	date only
commit message	full	full	top line
tags	yes	yes	yes
merges indicator	-	yes	-
status/delta	optional	optional	-
diff/patch	optional	optional	-
revision-id	optional	optional	-
branch nick	yes	-	-
foreign vcs properties	yes	yes	-
preferred levels	all	1	1
digital signature	optional	-	-

The default format is `long`. To change this, define the `log_format` setting in the `[DEFAULT]` section of `bazaar.conf` like this (say):

```
[DEFAULT]
log_format = short
```

Alternatively, to change the log format used for a given query, use the –long, –short or –line options.

If one of the standard log formats does not meet your needs, additional formats can be provided by plugins.

3.13 Other Storage Formats

Experimental formats are shown below.

development-colo (native) The 2a format with experimental support for colocated branches.

Deprecated formats are shown below.

pack-0.92 (native) Pack-based format used in 1.x series. Introduced in 0.92. Interoperates with bzr repositories before 0.92 but cannot be read by bzr < 0.92.

See *Storage Formats* for more about storage formats.

3.14 Revision Identifiers

A revision identifier refers to a specific state of a branch's history. It can be expressed in several ways. It can begin with a keyword to unambiguously specify a given lookup type; some examples are 'last:1', 'before:yesterday' and 'submit:'.

Alternately, it can be given without a keyword, in which case it will be checked as a revision number, a tag, a revision id, a date specification, or a branch specification, in that order. For example, 'date:today' could be written as simply 'today', though if you have a tag called 'today' that will be found first.

If 'REV1' and 'REV2' are revision identifiers, then 'REV1..REV2' denotes a revision range. Examples: '3647..3649', 'date:yesterday..-1' and 'branch:/path/to/branch1/..branch:/branch2' (note that there are no quotes or spaces around the '..').

Ranges are interpreted differently by different commands. To the "log" command, a range is a sequence of log messages, but to the "diff" command, the range denotes a change between revisions (and not a sequence of changes). In addition, "log" considers a closed range whereas "diff" and "merge" consider it to be open-ended, that is, they include one end but not the other. For example: "bzr log -r 3647..3649" shows the messages of revisions 3647, 3648 and 3649, while "bzr diff -r 3647..3649" includes the changes done in revisions 3648 and 3649, but not 3647.

The keywords used as revision selection methods are the following:

revid Selects a revision using the revision id.

submit Selects a common ancestor with the submit branch.

ancestor Selects a common ancestor with a second branch.

date Selects a revision on the basis of a datestamp.

branch Selects the last revision of a specified branch.

tag Selects a revision identified by a tag name.

revno Selects a revision using a number.

before Selects the parent of the revision specified.

annotate Select the revision that last modified the specified line.

mainline Select mainline revision that merged the specified revision.

last Selects the nth revision from the end.

In addition, plugins can provide other keywords.

A detailed description of each keyword is given below.

revid Supply a specific revision id, that can be used to specify any revision id in the ancestry of the branch. Including merges, and pending merges. Examples:

```
revid:aaaa@bbbb-123456789 -> Select revision 'aaaa@bbbb-123456789'
```

submit Diffing against this shows all the changes that were made in this branch, and is a good predictor of what merge will do. The submit branch is used by the bundle and merge directive commands. If no submit branch is specified, the parent branch is used instead.

The common ancestor is the last revision that existed in both branches. Usually this is the branch point, but it could also be a revision that was merged.

Examples:

```
$ bzr diff -r submit:
```

ancestor Supply the path to a branch to select the common ancestor.

The common ancestor is the last revision that existed in both branches. Usually this is the branch point, but it could also be a revision that was merged.

This is frequently used with 'diff' to return all of the changes that your branch introduces, while excluding the changes that you have not merged from the remote branch.

Examples:

```
ancestor:/path/to/branch
$ bzr diff -r ancestor:../../mainline/branch
```

date Supply a datestamp to select the first revision that matches the date. Date can be 'yesterday', 'today', 'tomorrow' or a YYYY-MM-DD string. Matches the first entry after a given date (either at midnight or at a specified time).

One way to display all the changes since yesterday would be:

```
bzr log -r date:yesterday..
```

Examples:

```
date:yesterday            -> select the first revision since yesterday
date:2006-08-14,17:10:14  -> select the first revision after
                             August 14th, 2006 at 5:10pm.
```

branch Supply the path to a branch to select its last revision.

Examples:

```
branch:/path/to/branch
```

tag Tags are stored in the branch and created by the 'tag' command.

revno Use an integer to specify a revision in the history of the branch. Optionally a branch can be specified. A negative number will count from the end of the branch (-1 is the last revision, -2 the previous one). If the negative number is larger than the branch's history, the first revision is returned. Examples:

```
revno:1                   -> return the first revision of this branch
revno:3:/path/to/branch   -> return the 3rd revision of
                             the branch '/path/to/branch'
revno:-1                  -> The last revision in a branch.
-2:http://other/branch    -> The second to last revision in the
                             remote branch.
-1000000                  -> Most likely the first revision, unless
                             your history is very long.
```

before Supply any revision spec to return the parent of that revision. This is mostly useful when inspecting revisions that are not in the revision history of a branch.

It is an error to request the parent of the null revision (before:0).

Examples:

```
before:1913    -> Return the parent of revno 1913 (revno 1912)
before:revid:aaaa@bbbb-1234567890  -> return the parent of revision
                                      aaaa@bbbb-1234567890
bzr diff -r before:1913..1913
      -> Find the changes between revision 1913 and its parent (1912).
         (What changes did revision 1913 introduce).
         This is equivalent to:  bzr diff -c 1913
```

annotate Select the revision that last modified the specified line. Line is specified as path:number. Path is a relative path to the file. Numbers start at 1, and are relative to the current version, not the last-committed version of the file.

mainline Select the revision that merged the specified revision into mainline.

last Supply a positive number to get the nth revision from the end. This is the same as supplying negative numbers to the 'revno:' spec. Examples:

```
last:1        -> return the last revision
last:3        -> return the revision 2 before the end.
```

3.15 Standard Options

Standard options are legal for all commands.

--help, -h	Show help message.
--verbose, -v	Display more information.
--quiet, -q	Only display errors and warnings.

Unlike global options, standard options can be used in aliases.

3.16 Status Flags

Status flags are used to summarise changes to the working tree in a concise manner. They are in the form:

```
xxx   <filename>
```

where the columns' meanings are as follows.

Column 1 - versioning/renames:

```
+ File versioned
- File unversioned
R File renamed
? File unknown
X File nonexistent (and unknown to bzr)
C File has conflicts
P Entry for a pending merge (not a file)
```

Column 2 - contents:

```
N File created
D File deleted
K File kind changed
M File modified
```

Column 3 - execute:

```
* The execute bit was changed
```

3.17 Special character handling in URLs

Bazaar allows locations to be specified in multiple ways, either:

- Fully qualified URLs
- File system paths, relative or absolute

Internally bzr treats all locations as URLs. For any file system paths that are specified it will automatically determine the appropriate URL representation, and escape special characters where necessary.

There are a few characters which have special meaning in URLs and need careful handling to avoid ambiguities. Characters can be escaped with a % and a hex value in URLs. Any non-ASCII characters in a file path will automatically be urlencoded when the path is converted to a URL.

URLs represent non-ASCII characters in an encoding defined by the server, but usually UTF-8. The % escapes should be of the UTF-8 bytes. Bazaar tries to be generous in what it accepts as a URL and to print them in a way that will be readable.

For example, if you have a directory named '/tmp/%2False' these are all valid ways of accessing the content (0x2F, or 47, is the ASCII code for forward slash):

```
cd /tmp
bzr log /tmp/%2False
bzr log %2False
bzr log file:///tmp/%252False
bzr log file://localhost/tmp/%252False
bzr log file:%252False
```

These are valid but do not refer to the same file:

```
bzr log file:///tmp/%2False (refers to a file called /tmp/\/alse)
bzr log %252False (refers to a file called /tmp/%252False)
```

Comma also has special meaning in URLs, because it denotes segment parameters

segment parameters: http://www.ietf.org/rfc/rfc3986.txt (section 3.3)

Comma is also special in any file system paths that are specified. To use a literal comma in a file system path, specify a URL and URL encode the comma:

```
bzr log foo,branch=bla # path "foo" with the segment parameter "branch" set to "bla"
bzr log file:foo%2Cbranch=bla # path "foo,branch=bla"
bzr log file:foo,branch=bla # path "foo" with segment parameter "branch" set to "bla"
```

3.18 URL Identifiers

Supported URL prefixes:

```
aftp://             Access using active FTP.
bzr://              Fast access using the Bazaar smart server.
bzr+ssh://          Fast access using the Bazaar smart server over SSH.
file://             Access using the standard filesystem (default)
ftp://              Access using passive FTP.
http://             Read-only access of branches exported on the web.
https://            Read-only access of branches exported on the web using SSL.
sftp://             Access using SFTP (most SSH servers provide SFTP).
```

Supported modifiers:

```
gio+                Access using any GIO supported protocols.
```

Bazaar supports all of the standard parts within the URL:

```
<protocol>://[user[:password]@]host[:port]/[path]
```

allowing URLs such as:

```
http://bzruser:BadPass@bzr.example.com:8080/bzr/trunk
```

For bzr+ssh:// and sftp:// URLs, Bazaar also supports paths that begin with '~' as meaning that the rest of the path should be interpreted relative to the remote user's home directory. For example if the user `remote` has a home directory of `/home/remote` on the server shell.example.com, then:

```
bzr+ssh://remote@shell.example.com/~/myproject/trunk
```

would refer to `/home/remote/myproject/trunk`.

Many commands that accept URLs also accept location aliases too. See *Location aliases* and *Special character handling in URLs*.

CHAPTER

FOUR

COMMANDS

4.1 add

Purpose Add specified files or directories.

Usage bzr add [FILE...]

Options

--dry-run	Show what would be done, but don't actually do anything.
-v, --verbose	Display more information.
--file-ids-from=ARG	Lookup file ids from this tree.
-N, --no-recurse	Don't recursively add the contents of directories.
-q, --quiet	Only display errors and warnings.
--usage	Show usage message and options.
-h, --help	Show help message.

Description In non-recursive mode, all the named items are added, regardless of whether they were previously ignored. A warning is given if any of the named files are already versioned.

In recursive mode (the default), files are treated the same way but the behaviour for directories is different. Directories that are already versioned do not give a warning. All directories, whether already versioned or not, are searched for files or subdirectories that are neither versioned or ignored, and these are added. This search proceeds recursively into versioned directories. If no names are given '.' is assumed.

A warning will be printed when nested trees are encountered, unless they are explicitly ignored.

Therefore simply saying 'bzr add' will version all files that are currently unknown.

Adding a file whose parent directory is not versioned will implicitly add the parent, and so on up to the root. This means you should never need to explicitly add a directory, they'll just get added when you add a file in the directory.

–dry-run will show which files would be added, but not actually add them.

–file-ids-from will try to use the file ids from the supplied path. It looks up ids trying to find a matching parent directory with the same filename, and then by pure path. This option is rarely needed but can be useful when adding the same logical file into two branches that will be merged later (without showing the two different adds as a conflict). It is also useful when merging another project into a subdirectory of this one.

Any files matching patterns in the ignore list will not be added unless they are explicitly mentioned.

In recursive mode, files larger than the configuration option add.maximum_file_size will be skipped. Named items are never skipped due to file size.

See also *ignore*, *remove*

4.2 alias

Purpose Set/unset and display aliases.

Usage bzr alias [NAME]

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
--remove	Remove the alias.
-h, --help	Show help message.

Examples Show the current aliases:

```
bzr alias
```

Show the alias specified for 'll':

```
bzr alias ll
```

Set an alias for 'll':

```
bzr alias ll="log --line -r-10..-1"
```

To remove an alias for 'll':

```
bzr alias --remove ll
```

4.3 annotate

Purpose Show the origin of each line in a file.

Usage bzr annotate FILENAME

Options

--all	Show annotations on all lines.
-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
--long	Show commit date in annotations.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
--show-ids	Show internal object ids.
-r ARG, --revision=ARG	See "help revisionspec" for details.

-h, --help Show help message.

Description This prints out the given file with an annotation on the left side indicating which revision, author and date introduced the change.

If the origin is the same for a run of consecutive lines, it is shown only at the top, unless the –all option is given.

Aliases ann, blame, praise

4.4 bind

Purpose Convert the current branch into a checkout of the supplied branch.

Usage bzr bind [LOCATION]

Options

--usage	Show usage message and options.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description If no branch is supplied, rebind to the last bound location.

Once converted into a checkout, commits must succeed on the master branch before they will be applied to the local branch.

Bound branches use the nickname of its master branch unless it is set locally, in which case binding will update the local nickname to be that of the master.

See also *checkouts*, *unbind*

4.5 branch

Purpose Create a new branch that is a copy of an existing branch.

Usage bzr branch FROM_LOCATION [TO_LOCATION]

Options

--use-existing-dir	By default branch will fail if the target directory exists, but does not already have a control directory. This flag will allow branch to proceed.
--stacked	Create a stacked branch referring to the source branch. The new branch will depend on the availability of the source branch for all operations.
-v, --verbose	Display more information.
--standalone	Do not use a shared repository, even if available.
--files-from=ARG	Get file contents from this tree.
-h, --help	Show help message.

-q, --quiet	Only display errors and warnings.
--switch	Switch the checkout in the current directory to the new branch.
--hardlink	Hard-link working tree files where possible.
--bind	Bind new branch to from location.
--usage	Show usage message and options.
--no-tree	Create a branch without a working-tree.
-r ARG, --revision=ARG	See "help revisionspec" for details.

Description If the TO_LOCATION is omitted, the last component of the FROM_LOCATION will be used. In other words, "branch ../foo/bar" will attempt to create ./bar. If the FROM_LOCATION has no / or path separator embedded, the TO_LOCATION is derived from the FROM_LOCATION by stripping a leading scheme or drive identifier, if any. For example, "branch lp:foo-bar" will attempt to create ./foo-bar.

To retrieve the branch as of a particular revision, supply the –revision parameter, as in "branch foo/bar -r 5".

The synonyms 'clone' and 'get' for this command are deprecated.

Aliases get, clone

See also *checkout*

4.6 branches

Purpose List the branches available at the current location.

Usage bzr branches [LOCATION]

Options

--usage	Show usage message and options.
-R, --recursive	Recursively scan for branches rather than just looking in the specified location.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description This command will print the names of all the branches at the current location.

4.7 break-lock

Purpose Break a dead lock.

Usage bzr break-lock [LOCATION]

Options

--force	Do not ask for confirmation before breaking the lock.
-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.

--usage	Show usage message and options.
--config	LOCATION is the directory where the config lock is.
-h, --help	Show help message.

Description This command breaks a lock on a repository, branch, working directory or config file.

CAUTION: Locks should only be broken when you are sure that the process holding the lock has been stopped.

You can get information on what locks are open via the 'bzr info [location]' command.

Examples bzr break-lock bzr break-lock bzr+ssh://example.com/bzr/foo bzr break-lock –conf ~/.bazaar

4.8 cat

Purpose Write the contents of a file as of a given revision to standard output.

Usage bzr cat FILENAME

Options

-v, --verbose	Display more information.
--name-from-revision	The path name in the old tree.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--filters	Apply content filters to display the convenience form.
--usage	Show usage message and options.
-r ARG, --revision=ARG	See "help revisionspec" for details.
-h, --help	Show help message.

Description If no revision is nominated, the last revision is used.

Note: Take care to redirect standard output when using this command on a binary file.

See also *ls*

4.9 check

Purpose Validate working tree structure, branch consistency and repository history.

Usage bzr check [PATH]

Options

-v, --verbose	Display more information.
--tree	Check the working tree related to the current directory.
-q, --quiet	Only display errors and warnings.
--repo	Check the repository related to the current directory.
--branch	Check the branch related to the current directory.
--usage	Show usage message and options.

-h, --help Show help message.

Description This command checks various invariants about branch and repository storage to detect data corruption or bzr bugs.

The working tree and branch checks will only give output if a problem is detected. The output fields of the repository check are:

revisions This is just the number of revisions checked. It doesn't indicate a problem.

versionedfiles This is just the number of versionedfiles checked. It doesn't indicate a problem.

unreferenced ancestors Texts that are ancestors of other texts, but are not properly referenced by the revision ancestry. This is a subtle problem that Bazaar can work around.

unique file texts This is the total number of unique file contents seen in the checked revisions. It does not indicate a problem.

repeated file texts This is the total number of repeated texts seen in the checked revisions. Texts can be repeated when their file entries are modified, but the file contents are not. It does not indicate a problem.

If no restrictions are specified, all Bazaar data that is found at the given location will be checked.

Examples Check the tree and branch at 'foo':

```
bzr check --tree --branch foo
```

Check only the repository at 'bar':

```
bzr check --repo bar
```

Check everything at 'baz':

```
bzr check baz
```

See also *reconcile*

4.10 checkout

Purpose Create a new checkout of an existing branch.

Usage bzr checkout [BRANCH_LOCATION] [TO_LOCATION]

Options

-v, --verbose	Display more information.
--files-from=ARG	Get file contents from this tree.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
--hardlink	Hard-link working tree files where possible.
--lightweight	Perform a lightweight checkout. Lightweight checkouts depend on access to the branch for every operation. Normal checkouts can perform common operations like diff and status without such access, and also support local commits.
--usage	Show usage message and options.
-r ARG, --revision=ARG	See "help revisionspec" for details.

Description If BRANCH_LOCATION is omitted, checkout will reconstitute a working tree for the branch found in '.'. This is useful if you have removed the working tree or if it was never created - i.e. if you pushed the branch to its current location using SFTP.

If the TO_LOCATION is omitted, the last component of the BRANCH_LOCATION will be used. In other words, "checkout ../foo/bar" will attempt to create ./bar. If the BRANCH_LOCATION has no / or path separator embedded, the TO_LOCATION is derived from the BRANCH_LOCATION by stripping a leading scheme or drive identifier, if any. For example, "checkout lp:foo-bar" will attempt to create ./foo-bar.

To retrieve the branch as of a particular revision, supply the –revision parameter, as in "checkout foo/bar -r 5". Note that this will be immediately out of date [so you cannot commit] but it may be useful (i.e. to examine old code.)

Aliases co

See also *branch*, *checkouts*, *remove-tree*, *working-trees*

4.11 clean-tree

Purpose Remove unwanted files from working tree.

Usage bzr clean-tree

Options

--ignored	Delete all ignored files.
--dry-run	Show files to delete instead of deleting them.
-v, --verbose	Display more information.
--unknown	Delete files unknown to bzr (default).
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
--force	Do not prompt before deleting.
--detritus	Delete conflict files, merge and revert backups, and failed self-test dirs.
-h, --help	Show help message.

Description By default, only unknown files, not ignored files, are deleted. Versioned files are never deleted.

Another class is 'detritus', which includes files emitted by bzr during normal operations and selftests. (The value of these files decreases with time.)

If no options are specified, unknown files are deleted. Otherwise, option flags are respected, and may be combined.

To check what clean-tree will do, use –dry-run.

4.12 commit

Purpose Commit changes into a new revision.

Usage bzr commit [SELECTED...]

Options

-v, --verbose	Display more information.
--author=ARG	Set the author's name, if it's different from the committer.
--commit-time=ARG	Manually set a commit time using commit date format, e.g. '2009-10-10 08:00:00 +0100'.
--unchanged	Commit even if nothing has changed.
--fixes=ARG	Mark a bug as being fixed by this revision (see "bzr help bugs").
-q, --quiet	Only display errors and warnings.
-p, --show-diff	When no message is supplied, show the diff along with the status summary in the message editor.
--strict	Refuse to commit if there are unknown files in the working tree.
--lossy	When committing to a foreign version control system do not push data that can not be natively represented.
-F MSGFILE, --file=MSGFILE	Take commit message from this file.
--usage	Show usage message and options.
-x ARG, --exclude=ARG	Do not consider changes made to a given path.
-m ARG, --message=ARG	Description of the new revision.
--local	Perform a local commit in a bound branch. Local commits are not pushed to the master branch until a normal commit is performed.
-h, --help	Show help message.

Description An explanatory message needs to be given for each commit. This is often done by using the –message option (getting the message from the command line) or by using the –file option (getting the message from a file). If neither of these options is given, an editor is opened for the user to enter the message. To see the changed files in the boilerplate text loaded into the editor, use the –show-diff option.

By default, the entire tree is committed and the person doing the commit is assumed to be the author. These defaults can be overridden as explained below.

Selective commits If selected files are specified, only changes to those files are committed. If a directory is specified then the directory and everything within it is committed.

When excludes are given, they take precedence over selected files. For example, to commit only changes within foo, but not changes within foo/bar:

```
bzr commit foo -x foo/bar
```

A selective commit after a merge is not yet supported.

Custom authors If the author of the change is not the same person as the committer, you can specify the author's name using the –author option. The name should be in the same format as a committer-id,

e.g. "John Doe <jdoe@example.com>". If there is more than one author of the change you can specify the option multiple times, once for each author.

Checks A common mistake is to forget to add a new file or directory before running the commit command. The –strict option checks for unknown files and aborts the commit if any are found. More advanced pre-commit checks can be implemented by defining hooks. See `bzr help hooks` for details.

Things to note If you accidentially commit the wrong changes or make a spelling mistake in the commit message say, you can use the uncommit command to undo it. See `bzr help uncommit` for details.

Hooks can also be configured to run after a commit. This allows you to trigger updates to external systems like bug trackers. The –fixes option can be used to record the association between a revision and one or more bugs. See `bzr help bugs` for details.

Aliases ci, checkin

See also *add*, *bugs*, *hooks*, *uncommit*

4.13 config

Purpose Display, set or remove a configuration option.

Usage bzr config [NAME]

Options

--all	Display all the defined values for the matching options.
-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
--remove	Remove the option from the configuration file.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
--scope=ARG	Reduce the scope to the specified configuration file.
-h, --help	Show help message.

Description Display the active value for option NAME.

If –all is specified, NAME is interpreted as a regular expression and all matching options are displayed mentioning their scope and without resolving option references in the value). The active value that bzr will take into account is the first one displayed for each option.

If NAME is not given, –all .* is implied (all options are displayed for the current scope).

Setting a value is achieved by using NAME=value without spaces. The value is set in the most relevant scope and can be checked by displaying the option again.

Removing a value is achieved by using –remove NAME.

See also *configuration*

4.14 conflicts

Purpose List files with conflicts.

Usage bzr conflicts

Options

-v, --verbose	Display more information.
--text	List paths of files with text conflicts.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
-h, --help	Show help message.

Description Merge will do its best to combine the changes in two branches, but there are some kinds of problems only a human can fix. When it encounters those, it will mark a conflict. A conflict means that you need to fix something, before you can commit.

Conflicts normally are listed as short, human-readable messages. If –text is supplied, the pathnames of files with text conflicts are listed, instead. (This is useful for editing all files with text conflicts.)

Use bzr resolve when you have fixed a problem.

See also *conflict-types*, *resolve*

4.15 deleted

Purpose List files deleted in the working tree.

Usage bzr deleted

Options

-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
--show-ids	Show internal object ids.
-h, --help	Show help message.

See also *ls*, *status*

4.16 diff

Purpose Show differences in the working tree, between revisions or branches.

Usage bzr diff [FILE...]

Options

--old=ARG	Branch/tree to compare from.

-v, --verbose Display more information.

-F ARG, --format=ARG Diff format to use.

-q, --quiet Only display errors and warnings.

-p ARG, --prefix=ARG Set prefixes added to old and new filenames, as two values separated by a colon. (eg "old/:new/").

--context=ARG How many lines of context to show.

--using=ARG Use this command to compare files.

--usage Show usage message and options.

--new=ARG Branch/tree to compare to.

-r ARG, --revision=ARG See "help revisionspec" for details.

--diff-options=ARG Pass these options to the external diff program.

-c ARG, --change=ARG Select changes introduced by the specified revision. See also "help revisionspec".

-h, --help Show help message.

Description If no arguments are given, all changes for the current tree are listed. If files are given, only the changes in those files are listed. Remote and multiple branches can be compared by using the –old and –new options. If not provided, the default for both is derived from the first argument, if any, or the current tree if no arguments are given.

"bzr diff -p1" is equivalent to "bzr diff –prefix old/:new/", and produces patches suitable for "patch -p1".

Note that when using the -r argument with a range of revisions, the differences are computed between the two specified revisions. That is, the command does not show the changes introduced by the first revision in the range. This differs from the interpretation of revision ranges used by "bzr log" which includes the first revision in the range.

Exit values 1 - changed 2 - unrepresentable changes 3 - error 0 - no change

Examples Shows the difference in the working tree versus the last commit:

```
bzr diff
```

Difference between the working tree and revision 1:

```
bzr diff -r1
```

Difference between revision 3 and revision 1:

```
bzr diff -r1..3
```

Difference between revision 3 and revision 1 for branch xxx:

```
bzr diff -r1..3 xxx
```

The changes introduced by revision 2 (equivalent to -r1..2):

```
bzr diff -c2
```

To see the changes introduced by revision X:

```
bzr diff -cX
```

Note that in the case of a merge, the -c option shows the changes compared to the left hand parent. To see the changes against another parent, use:

```
bzr diff -r<chosen_parent>..X
```

The changes between the current revision and the previous revision (equivalent to -c-1 and -r-2..-1)

bzr diff -r-2..

Show just the differences for file NEWS:

```
bzr diff NEWS
```

Show the differences in working tree xxx for file NEWS:

```
bzr diff xxx/NEWS
```

Show the differences from branch xxx to this working tree:

bzr diff –old xxx

Show the differences between two branches for file NEWS:

```
bzr diff --old xxx --new yyy NEWS
```

Same as 'bzr diff' but prefix paths with old/ and new/:

```
bzr diff --prefix old/:new/
```

Show the differences using a custom diff program with options:

```
bzr diff --using /usr/bin/diff --diff-options -wu
```

Aliases di, dif

See also *status*

4.17 dpush

Purpose Push into a different VCS without any custom bzr metadata.

Usage bzr dpush [LOCATION]

Options

--remember	Remember the specified location as a default.
--strict	Refuse to push if there are uncommitted changes in the working tree, –no-strict disables the check.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to push from, rather than the one containing the working directory.
--usage	Show usage message and options.
--no-rebase	Do not rebase after push.
-v, --verbose	Display more information.

Description This will afterwards rebase the local branch on the remote branch unless the –no-rebase option is used, in which case the two branches will be out of sync after the push.

4.18 export

Purpose Export current or past revision to a destination directory or archive.

Usage bzr export DEST [BRANCH_OR_SUBDIR]

Options

-v, --verbose	Display more information.
--format=ARG	Type of file to export to.
-q, --quiet	Only display errors and warnings.
--per-file-timestamps	Set modification time of files to that of the last revision in which it was changed.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--uncommitted	Export the working tree contents rather than that of the last revision.
--filters	Apply content filters to export the convenient form.
--usage	Show usage message and options.
-r ARG, --revision=ARG	See "help revisionspec" for details.
--root=ARG	Name of the root directory inside the exported file.
-h, --help	Show help message.

Description If no revision is specified this exports the last committed revision.

Format may be an "exporter" name, such as tar, tgz, tbz2. If none is given, try to find the format with the extension. If no extension is found exports to a directory (equivalent to –format=dir).

If root is supplied, it will be used as the root directory inside container formats (tar, zip, etc). If it is not supplied it will default to the exported filename. The root option has no effect for 'dir' format.

If branch is omitted then the branch containing the current working directory will be used.

Note: Export of tree with non-ASCII filenames to zip is not supported.

Supported formats	Autodetected by extension
dir	(none)
tar	.tar
tbz2	.tar.bz2, .tbz2
tgz	.tar.gz, .tgz
zip	.zip

4.19 help

Purpose Show help on a command or other topic.

Usage bzr help [TOPIC]

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
--long	Show help on all commands.
-h, --help	Show help message.

Aliases ?, –help, -?, -h

See also topics

4.20 ignore

Purpose Ignore specified files or patterns.

Usage bzr ignore [NAME_PATTERN...]

Options

--default-rules	Display the default ignore rules that bzr uses.
-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
-h, --help	Show help message.

Description See `bzr help patterns` for details on the syntax of patterns.

If a .bzrignore file does not exist, the ignore command will create one and add the specified files or patterns to the newly created file. The ignore command will also automatically add the .bzrignore file to be versioned. Creating a .bzrignore file without the use of the ignore command will require an explicit add command.

To remove patterns from the ignore list, edit the .bzrignore file. After adding, editing or deleting that file either indirectly by using this command or directly by using an editor, be sure to commit it.

Bazaar also supports a global ignore file ~/.bazaar/ignore. On Windows the global ignore file can be found in the application data directory as C:Documents and Settings<user>Application DataBazaar2.0ignore. Global ignores are not touched by this command. The global ignore file can be edited directly using an editor.

Patterns prefixed with '!' are exceptions to ignore patterns and take precedence over regular ignores. Such exceptions are used to specify files that should be versioned which would otherwise be ignored.

Patterns prefixed with '!!' act as regular ignore patterns, but have precedence over the '!' exception patterns.

Notes

- Ignore patterns containing shell wildcards must be quoted from the shell on Unix.
- Ignore patterns starting with "#" act as comments in the ignore file. To ignore patterns that begin with that character, use the "RE:" prefix.

Examples Ignore the top level Makefile:

```
bzr ignore ./Makefile
```

Ignore .class files in all directories...:

```
bzr ignore "*.class"
```

...but do not ignore "special.class":

```
bzr ignore "!special.class"
```

Ignore files whose name begins with the "#" character:

```
bzr ignore "RE:^#"
```

Ignore .o files under the lib directory:

```
bzr ignore "lib/**/*.o"
```

Ignore .o files under the lib directory:

```
bzr ignore "RE:lib/.*\.o"
```

Ignore everything but the "debian" toplevel directory:

```
bzr ignore "RE:(?!debian/).*"
```

Ignore everything except the "local" toplevel directory, but always ignore autosave files ending in ~, even under local/:

```
bzr ignore "*"
bzr ignore "!./local"
bzr ignore "!!*~"
```

See also *ignored*, *patterns*, *status*

4.21 ignored

Purpose List ignored files and the patterns that matched them.

Usage bzr ignored

Options

--usage	Show usage message and options.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description List all the ignored files and the ignore pattern that caused the file to be ignored.

Alternatively, to list just the files:

```
bzr ls --ignored
```

See also *ignore*, *ls*

4.22 info

Purpose Show information about a working tree, branch or repository.

Usage bzr info [LOCATION]

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description This command will show all known locations and formats associated to the tree, branch or repository.

In verbose mode, statistical information is included with each report. To see extended statistic information, use a verbosity level of 2 or higher by specifying the verbose option multiple times, e.g. -vv.

Branches and working trees will also report any missing revisions.

Examples Display information on the format and related locations:

bzr info

Display the above together with extended format information and basic statistics (like the number of files in the working tree and number of revisions in the branch and repository):

bzr info -v

Display the above together with number of committers to the branch:

bzr info -vv

See also *repositories*, *revno*, *working-trees*

4.23 init

Purpose Make a directory into a versioned branch.

Usage bzr init [LOCATION]

Options

-v, --verbose	Display more information.
--create-prefix	Create the path leading up to the branch if it does not already exist.
-q, --quiet	Only display errors and warnings.
--append-revisions-only	Never change revnos or the existing log. Append revisions to it only.
--usage	Show usage message and options.
--no-tree	Create a branch without a working tree.
-h, --help	Show help message.

Branch format:

--format=ARG Specify a format for this branch. See "help formats".

--2a Format for the bzr 2.0 series. Uses group-compress storage. Provides rich roots which are a one-way transition.

--default Format for the bzr 2.0 series. Uses group-compress storage. Provides rich roots which are a one-way transition.

--development-colo The 2a format with experimental support for colocated branches.

–pack-0.92 Pack-based format used in 1.x series. Introduced in 0.92. Interoperates with bzr repositories before 0.92 but cannot be read by bzr < 0.92.

Description Use this to create an empty branch, or before importing an existing project.

If there is a repository in a parent directory of the location, then the history of the branch will be stored in the repository. Otherwise init creates a standalone branch which carries its own history in the .bzr directory.

If there is already a branch at the location but it has no working tree, the tree can be populated with 'bzr checkout'.

Recipe for importing a tree of files:

```
cd ~/project
bzr init
bzr add .
bzr status
bzr commit -m "imported project"
```

See also *branch*, *checkout*, *init-repository*

4.24 init-repository

Purpose Create a shared repository for branches to share storage space.

Usage bzr init-repository LOCATION

Options

--no-trees Branches in the repository will default to not having a working tree.

-v, --verbose Display more information.

-q, --quiet Only display errors and warnings.

--usage Show usage message and options.

-h, --help Show help message.

Repository format:

--format=ARG Specify a format for this repository. See "bzr help formats" for details.

--2a Format for the bzr 2.0 series. Uses group-compress storage. Provides rich roots which are a one-way transition.

--default Format for the bzr 2.0 series. Uses group-compress storage. Provides rich roots which are a one-way transition.

--development-colo The 2a format with experimental support for colocated branches.

–pack-0.92 Pack-based format used in 1.x series. Introduced in 0.92. Interoperates with bzr repositories before 0.92 but cannot be read by bzr < 0.92.

Description New branches created under the repository directory will store their revisions in the repository, not in the branch directory. For branches with shared history, this reduces the amount of storage needed and speeds up the creation of new branches.

If the –no-trees option is given then the branches in the repository will not have working trees by default. They will still exist as directories on disk, but they will not have separate copies of the files at a certain revision. This can be useful for repositories that store branches which are interacted with through checkouts or remote branches, such as on a server.

Examples Create a shared repository holding just branches:

```
bzr init-repo --no-trees repo
bzr init repo/trunk
```

Make a lightweight checkout elsewhere:

```
bzr checkout --lightweight repo/trunk trunk-checkout
cd trunk-checkout
(add files here)
```

Aliases init-repo

See also *branch*, *checkout*, *init*, *repositories*

4.25 join

Purpose Combine a tree into its containing tree.

Usage bzr join TREE

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description This command requires the target tree to be in a rich-root format.

The TREE argument should be an independent tree, inside another tree, but not part of it. (Such trees can be produced by “bzr split”, but also by running “bzr branch” with the target inside a tree.)

The result is a combined tree, with the subtree no longer an independent part. This is marked as a merge of the subtree into the containing tree, and all history is preserved.

See also *split*

4.26 log

Purpose Show historical log for a branch or subset of a branch.

Usage bzr log [FILE...]

Options

--signatures Show digital signature validity.

-v, --verbose Show files changed in each revision.

--include-merged Show merged revisions like –levels 0 does.

--timezone=ARG Display timezone as local, original, or utc.

-h, --help Show help message.

-p, --show-diff Show changes made in each revision as a patch.

--forward Show from oldest to newest.

-m ARG, --match=ARG Show revisions whose properties match this expression.

-r ARG, --revision=ARG See "help revisionspec" for details.

--omit-merges Do not report commits with more than one parent.

--usage Show usage message and options.

-n N, --levels=N Number of levels to display - 0 for all, 1 for flat.

--authors=ARG What names to list as authors - first, all or committer.

--match-author=ARG Show revisions whose authors match this expression.

--match-message=ARG Show revisions whose message matches this expression.

-c ARG, --change=ARG Show just the specified revision. See also "help revisionspec".

--match-bugs=ARG Show revisions whose bugs match this expression.

-q, --quiet Only display errors and warnings.

--match-committer=ARG Show revisions whose committer matches this expression.

-l N, --limit=N Limit the output to the first N revisions.

--show-ids Show internal object ids.

--exclude-common-ancestry Display only the revisions that are not part of both ancestries (require -rX..Y).

Log format:

--log-format=ARG Use specified log format.

--gnu-changelog Format used by GNU ChangeLog files.

--line Log format with one line per revision.

--long Detailed log format.

-S, --short Moderately short log format.

Description log is bzr's default tool for exploring the history of a branch. The branch to use is taken from the first parameter. If no parameters are given, the branch containing the working directory is logged. Here are some simple examples:

```
bzr log                       log the current branch
bzr log foo.py                log a file in its branch
bzr log http://server/branch  log a branch on a server
```

The filtering, ordering and information shown for each revision can be controlled as explained below. By default, all revisions are shown sorted (topologically) so that newer revisions appear before older ones and descendants always appear before ancestors. If displayed, merged revisions are shown indented under the revision in which they were merged.

Output control The log format controls how information about each revision is displayed. The standard log formats are called `long`, `short` and `line`. The default is long. See `bzr help log-formats` for more details on log formats.

The following options can be used to control what information is displayed:

```
-l N        display a maximum of N revisions
-n N        display N levels of revisions (0 for all, 1 for collapsed)
-v          display a status summary (delta) for each revision
-p          display a diff (patch) for each revision
--show-ids  display revision-ids (and file-ids), not just revnos
```

Note that the default number of levels to display is a function of the log format. If the -n option is not used, the standard log formats show just the top level (mainline).

Status summaries are shown using status flags like A, M, etc. To see the changes explained using words like `added` and `modified` instead, use the -vv option.

Ordering control To display revisions from oldest to newest, use the –forward option. In most cases, using this option will have little impact on the total time taken to produce a log, though –forward does not incrementally display revisions like –reverse does when it can.

Revision filtering The -r option can be used to specify what revision or range of revisions to filter against. The various forms are shown below:

```
-rX      display revision X
-rX..    display revision X and later
-r..Y    display up to and including revision Y
-rX..Y   display from X to Y inclusive
```

See `bzr help revisionspec` for details on how to specify X and Y. Some common examples are given below:

```
-r-1                show just the tip
-r-10..             show the last 10 mainline revisions
-rsubmit:..         show what's new on this branch
-rancestor:path..   show changes since the common ancestor of this
                    branch and the one at location path
-rdate:yesterday..  show changes since yesterday
```

When logging a range of revisions using -rX..Y, log starts at revision Y and searches back in history through the primary ("left-hand") parents until it finds X. When logging just the top level (using -n1), an error is reported if X is not found along the way. If multi-level logging is used (-n0), X may be a nested merge revision and the log will be truncated accordingly.

Path filtering If parameters are given and the first one is not a branch, the log will be filtered to show only those revisions that changed the nominated files or directories.

Filenames are interpreted within their historical context. To log a deleted file, specify a revision range so that the file existed at the end or start of the range.

Historical context is also important when interpreting pathnames of renamed files/directories. Consider the following example:

- revision 1: add tutorial.txt
- revision 2: modify tutorial.txt
- revision 3: rename tutorial.txt to guide.txt; add tutorial.txt

In this case:

- `bzr log guide.txt` will log the file added in revision 1
- `bzr log tutorial.txt` will log the new file added in revision 3
- `bzr log -r2 -p tutorial.txt` will show the changes made to the original file in revision 2.
- `bzr log -r2 -p guide.txt` will display an error message as there was no file called guide.txt in revision 2.

Renames are always followed by log. By design, there is no need to explicitly ask for this (and no way to stop logging a file back until it was last renamed).

Other filtering The –match option can be used for finding revisions that match a regular expression in a commit message, committer, author or bug. Specifying the option several times will match any of the supplied expressions. –match-author, –match-bugs, –match-committer and –match-message can be used to only match a specific field.

Tips & tricks GUI tools and IDEs are often better at exploring history than command line tools: you may prefer qlog or viz from qbzr or bzr-gtk, the bzr-explorer shell, or the Loggerhead web interface. See the Plugin Guide <http://doc.bazaar.canonical.com/plugins/en/> and <http://wiki.bazaar.canonical.com/IDEIntegration>.

You may find it useful to add the aliases below to `bazaar.conf`:

```
[ALIASES]
tip = log -r-1
top = log -l10 --line
show = log -v -p
```

`bzr tip` will then show the latest revision while `bzr top` will show the last 10 mainline revisions. To see the details of a particular revision X, `bzr show -rX`.

If you are interested in looking deeper into a particular merge X, use `bzr log -n0 -rX`.

`bzr log -v` on a branch with lots of history is currently very slow. A fix for this issue is currently under development. With or without that fix, it is recommended that a revision range be given when using the -v option.

bzr has a generic full-text matching plugin, bzr-search, that can be used to find revisions matching user names, commit messages, etc. Among other features, this plugin can find all revisions containing a list of words but not others.

When exploring non-mainline history on large projects with deep history, the performance of log can be greatly improved by installing the historycache plugin. This plugin buffers historical information trading disk space for faster speed.

See also *log-formats*, *revisionspec*

4.27 ls

Purpose List files in a tree.

Usage bzr ls [PATH]

Options

--from-root	Print paths relative to the root of the branch.
-i, --ignored	Print ignored files.
-k ARG, --kind=ARG	List entries of a particular kind: file, directory, symlink.
-v, --verbose	Display more information.
-R, --recursive	Recurse into subdirectories.
-V, --versioned	Print versioned files.
-u, --unknown	Print unknown files.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
--show-ids	Show internal object ids.
-0, --null	Use an ASCII NUL (0) separator rather than a newline.
-r ARG, --revision=ARG	See "help revisionspec" for details.

See also *cat*, *status*

4.28 merge

Purpose Perform a three-way merge.

Usage bzr merge [LOCATION]

Options

--pull	If the destination is already completely merged into the source, pull from the source rather than merging. When this happens, you do not need to commit the result.
--remember	Remember the specified location as a default.
-i, --interactive	Select changes interactively.
--force	Merge even if the destination tree has uncommitted changes.
-v, --verbose	Display more information.
--reprocess	Reprocess to reduce spurious conflicts.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to merge into, rather than the one containing the working directory.

--uncommitted Apply uncommitted changes from a working copy, instead of branch changes.

--usage Show usage message and options.

--show-base Show base revision text in conflicts.

--preview Instead of merging, show a diff of the merge.

-c ARG, --change=ARG Select changes introduced by the specified revision. See also "help revisionspec".

-r ARG, --revision=ARG See "help revisionspec" for details.

Merge algorithm:

--merge-type=ARG Select a particular merge algorithm.

--diff3 Merge using external diff3.

--lca LCA-newness merge.

--merge3 Native diff3-style merge.

--weave Weave-based merge.

Description The source of the merge can be specified either in the form of a branch, or in the form of a path to a file containing a merge directive generated with bzr send. If neither is specified, the default is the upstream branch or the branch most recently merged using –remember. The source of the merge may also be specified in the form of a path to a file in another branch: in this case, only the modifications to that file are merged into the current working tree.

When merging from a branch, by default bzr will try to merge in all new work from the other branch, automatically determining an appropriate base revision. If this fails, you may need to give an explicit base.

To pick a different ending revision, pass "–revision OTHER". bzr will try to merge in all new work up to and including revision OTHER.

If you specify two values, "–revision BASE..OTHER", only revisions BASE through OTHER, excluding BASE but including OTHER, will be merged. If this causes some revisions to be skipped, i.e. if the destination branch does not already contain revision BASE, such a merge is commonly referred to as a "cherrypick". Unlike a normal merge, Bazaar does not currently track cherrypicks. The changes look like a normal commit, and the history of the changes from the other branch is not stored in the commit.

Revision numbers are always relative to the source branch.

Merge will do its best to combine the changes in two branches, but there are some kinds of problems only a human can fix. When it encounters those, it will mark a conflict. A conflict means that you need to fix something, before you can commit.

Use bzr resolve when you have fixed a problem. See also bzr conflicts.

If there is no default branch set, the first merge will set it (use –no-remember to avoid setting it). After that, you can omit the branch to use the default. To change the default, use –remember. The value will only be saved if the remote location can be accessed.

The results of the merge are placed into the destination working directory, where they can be reviewed (with bzr diff), tested, and then committed to record the result of the merge.

merge refuses to run if there are any uncommitted changes, unless –force is given. If –force is given, then the changes from the source will be merged with the current working tree, including

any uncommitted changes in the tree. The –force option can also be used to create a merge revision which has more than two parents.

If one would like to merge changes from the working tree of the other branch without merging any committed revisions, the –uncommitted option can be given.

To select only some changes to merge, use "merge -i", which will prompt you to apply each diff hunk and file change, similar to "shelve".

Examples To merge all new revisions from bzr.dev:

```
bzr merge ../bzr.dev
```

To merge changes up to and including revision 82 from bzr.dev:

```
bzr merge -r 82 ../bzr.dev
```

To merge the changes introduced by 82, without previous changes:

```
bzr merge -r 81..82 ../bzr.dev
```

To apply a merge directive contained in /tmp/merge:

```
bzr merge /tmp/merge
```

To create a merge revision with three parents from two branches feature1a and feature1b:

bzr merge ../feature1a bzr merge ../feature1b –force bzr commit -m 'revision with three parents'

See also *remerge*, *send*, *status-flags*, *update*

4.29 missing

Purpose Show unmerged/unpulled revisions between two branches.

Usage bzr missing [OTHER_BRANCH]

Options

--include-merged	Show all revisions in addition to the mainline ones.
-v, --verbose	Display more information.
--this	Same as –mine-only.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--other	Same as –theirs-only.
--mine-only	Display changes in the local branch only.
--usage	Show usage message and options.
--my-revision=ARG	Filter on local branch revisions (inclusive). See "help revisionspec" for details.
--show-ids	Show internal object ids.

-r ARG, --revision=ARG Filter on other branch revisions (inclusive). See "help revisionspec" for details.

--theirs-only Display changes in the remote branch only.

--reverse Reverse the order of revisions.

Log format:

--log-format=ARG Use specified log format.

--gnu-changelog Format used by GNU ChangeLog files.

--line Log format with one line per revision.

--long Detailed log format.

-S, --short Moderately short log format.

Description OTHER_BRANCH may be local or remote.

To filter on a range of revisions, you can use the command -r begin..end -r revision requests a specific revision, -r ..end or -r begin.. are also valid.

Exit values 1 - some missing revisions 0 - no missing revisions

Examples Determine the missing revisions between this and the branch at the remembered pull location:

```
bzr missing
```

Determine the missing revisions between this and another branch:

```
bzr missing http://server/branch
```

Determine the missing revisions up to a specific revision on the other branch:

```
bzr missing -r ..-10
```

Determine the missing revisions up to a specific revision on this branch:

```
bzr missing --my-revision ..-10
```

See also *merge*, *pull*

4.30 mkdir

Purpose Create a new versioned directory.

Usage bzr mkdir DIR...

Options

--usage Show usage message and options.

-q, --quiet Only display errors and warnings.

-p, --parents No error if existing, make parent directories as needed.

-v, --verbose Display more information.

-h, --help Show help message.

Description This is equivalent to creating the directory and then adding it.

4.31 mv

Purpose Move or rename a file.

Usage bzr mv OLDNAME NEWNAME

bzr mv SOURCE... DESTINATION

Options

--dry-run	Avoid making changes when guessing renames.
-v, --verbose	Display more information.
--auto	Automatically guess renames.
--after	Move only the bzr identifier of the file, because the file has already been moved.
-q, --quiet	Only display errors and warnings.
--usage	Show usage message and options.
-h, --help	Show help message.

Description If the last argument is a versioned directory, all the other names are moved into it. Otherwise, there must be exactly two arguments and the file is changed to a new name.

If OLDNAME does not exist on the filesystem but is versioned and NEWNAME does exist on the filesystem but is not versioned, mv assumes that the file has been manually moved and only updates its internal inventory to reflect that change. The same is valid when moving many SOURCE files to a DESTINATION.

Files cannot be moved between branches.

Aliases move, rename

4.32 nick

Purpose Print or set the branch nickname.

Usage bzr nick [NICKNAME]

Options

--usage	Show usage message and options.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description If unset, the colocated branch name is used for colocated branches, and the branch directory name is used for other branches. To print the current nickname, execute with no argument.

Bound branches use the nickname of its master branch unless it is set locally.

See also *info*

4.33 pack

Purpose Compress the data within a repository.

Usage bzr pack [BRANCH_OR_REPO]

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
--clean-obsolete-packs	Delete obsolete packs to save disk space.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description This operation compresses the data within a bazaar repository. As bazaar supports automatic packing of repository, this operation is normally not required to be done manually.

During the pack operation, bazaar takes a backup of existing repository data, i.e. pack files. This backup is eventually removed by bazaar automatically when it is safe to do so. To save disk space by removing the backed up pack files, the –clean-obsolete-packs option may be used.

Warning: If you use –clean-obsolete-packs and your machine crashes during or immediately after repacking, you may be left with a state where the deletion has been written to disk but the new packs have not been. In this case the repository may be unusable.

See also *repositories*

4.34 ping

Purpose Pings a Bazaar smart server.

Usage bzr ping LOCATION

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description This command sends a ‘hello’ request to the given location using the bzr smart protocol, and reports the response.

4.35 plugins

Purpose List the installed plugins.

Usage bzr plugins

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.

-v, --verbose Display more information.

-h, --help Show help message.

Description This command displays the list of installed plugins including version of plugin and a short description of each.

–verbose shows the path where each plugin is located.

A plugin is an external component for Bazaar that extends the revision control system, by adding or replacing code in Bazaar. Plugins can do a variety of things, including overriding commands, adding new commands, providing additional network transports and customizing log output.

See the Bazaar Plugin Guide <http://doc.bazaar.canonical.com/plugins/en/> for further information on plugins including where to find them and how to install them. Instructions are also provided there on how to write new plugins using the Python programming language.

4.36 pull

Purpose Turn this branch into a mirror of another branch.

Usage bzr pull [LOCATION]

Options

--remember Remember the specified location as a default.

--overwrite-tags Overwrite tags only.

-h, --help Show help message.

-q, --quiet Only display errors and warnings.

-d ARG, --directory=ARG Branch to pull into, rather than the one containing the working directory.

--usage Show usage message and options.

--show-base Show base revision text in conflicts.

-r ARG, --revision=ARG See “help revisionspec” for details.

--local Perform a local pull in a bound branch. Local pulls are not applied to the master branch.

--overwrite Ignore differences between branches and overwrite unconditionally.

-v, --verbose Show logs of pulled revisions.

Description By default, this command only works on branches that have not diverged. Branches are considered diverged if the destination branch’s most recent commit is one that has not been merged (directly or indirectly) into the parent.

If branches have diverged, you can use ‘bzr merge’ to integrate the changes from one into the other. Once one branch has merged, the other should be able to pull it again.

If you want to replace your local changes and just want your branch to match the remote one, use pull –overwrite. This will work even if the two branches have diverged.

If there is no default location set, the first pull will set it (use –no-remember to avoid setting it). After that, you can omit the location to use the default. To change the default, use –remember. The value will only be saved if the remote location can be accessed.

The –verbose option will display the revisions pulled using the log_format configuration option. You can use a different format by overriding it with -Olog_format=<other_format>.

Note: The location can be specified either in the form of a branch, or in the form of a path to a file containing a merge directive generated with bzr send.

See also *push*, *send*, *status-flags*, *update*

4.37 push

Purpose Update a mirror of this branch.

Usage bzr push [LOCATION]

Options

--stacked	Create a stacked branch that references the public location of the parent branch.
--remember	Remember the specified location as a default.
--strict	Refuse to push if there are uncommitted changes in the working tree, –no-strict disables the check.
--create-prefix	Create the path leading up to the branch if it does not already exist.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
--stacked-on=ARG	Create a stacked branch that refers to another branch for the commit history. Only the work not present in the referenced branch is included in the branch created.
-d ARG, --directory=ARG	Branch to push from, rather than the one containing the working directory.
--use-existing-dir	By default push will fail if the target directory exists, but does not already have a control directory. This flag will allow push to proceed.
--usage	Show usage message and options.
--no-tree	Don't populate the working tree, even for protocols that support it.
--overwrite-tags	Overwrite tags only.
-r ARG, --revision=ARG	See "help revisionspec" for details.
--overwrite	Ignore differences between branches and overwrite unconditionally.
-v, --verbose	Display more information.

Description The target branch will not have its working tree populated because this is both expensive, and is not supported on remote file systems.

Some smart servers or protocols *may* put the working tree in place in the future.

This command only works on branches that have not diverged. Branches are considered diverged if the destination branch's most recent commit is one that has not been merged (directly or indirectly) by the source branch.

If branches have diverged, you can use 'bzr push –overwrite' to replace the other branch completely, discarding its unmerged changes.

If you want to ensure you have the different changes in the other branch, do a merge (see bzr help merge) from the other branch, and commit that. After that you will be able to do a push without '–overwrite'.

If there is no default push location set, the first push will set it (use –no-remember to avoid setting it). After that, you can omit the location to use the default. To change the default, use –remember. The value will only be saved if the remote location can be accessed.

The –verbose option will display the revisions pushed using the log_format configuration option. You can use a different format by overriding it with -Olog_format=<other_format>.

See also *pull*, *update*, *working-trees*

4.38 reconcile

Purpose Reconcile bzr metadata in a branch.

Usage bzr reconcile [BRANCH]

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description This can correct data mismatches that may have been caused by previous ghost operations or bzr upgrades. You should only need to run this command if 'bzr check' or a bzr developer advises you to run it.

If a second branch is provided, cross-branch reconciliation is also attempted, which will check that data like the tree root id which was not present in very early bzr versions is represented correctly in both branches.

At the same time it is run it may recompress data resulting in a potential saving in disk space or performance gain.

The branch *MUST* be on a listable system such as local disk or sftp.

See also *check*

4.39 reconfigure

Purpose Reconfigure the type of a bzr directory.

Usage bzr reconfigure [LOCATION]

Options

--force	Perform reconfiguration even if local changes will be lost.

-v, --verbose Display more information.

--unstacked Reconfigure a branch to be unstacked. This may require copying substantial data into it.

-q, --quiet Only display errors and warnings.

--stacked-on=ARG Reconfigure a branch to be stacked on another branch.

--usage Show usage message and options.

--bind-to=ARG Branch to bind checkout to.

-h, --help Show help message.

Tree type:

--branch Reconfigure to be an unbound branch with no working tree.

--checkout Reconfigure to be a bound branch with a working tree.

--lightweight-checkout Reconfigure to be a lightweight checkout (with no local history).

--tree Reconfigure to be an unbound branch with a working tree.

Trees in Repository:

--with-no-trees Reconfigure repository to not create working trees on branches by default.

--with-trees Reconfigure repository to create working trees on branches by default.

Repository type:

--standalone Reconfigure to be a standalone branch (i.e. stop using shared repository).

--use-shared Reconfigure to use a shared repository.

Description A target configuration must be specified.

For checkouts, the bind-to location will be auto-detected if not specified. The order of preference is 1. For a lightweight checkout, the current bound location. 2. For branches that used to be checkouts, the previously-bound location. 3. The push location. 4. The parent location. If none of these is available, –bind-to must be specified.

See also *branches*, *checkouts*, *standalone-trees*, *working-trees*

4.40 remerge

Purpose Redo a merge.

Usage bzr remerge [FILE...]

Options

-v, --verbose Display more information.

--reprocess Reprocess to reduce spurious conflicts.

-q, --quiet Only display errors and warnings.

--usage	Show usage message and options.
--show-base	Show base revision text in conflicts.
-h, --help	Show help message.

Merge algorithm:

--merge-type=ARG	Select a particular merge algorithm.
--diff3	Merge using external diff3.
--lca	LCA-newness merge.
--merge3	Native diff3-style merge.
--weave	Weave-based merge.

Description Use this if you want to try a different merge technique while resolving conflicts. Some merge techniques are better than others, and remerge lets you try different ones on different files.

The options for remerge have the same meaning and defaults as the ones for merge. The difference is that remerge can (only) be run when there is a pending merge, and it lets you specify particular files.

Examples Re-do the merge of all conflicted files, and show the base text in conflict regions, in addition to the usual THIS and OTHER texts:

```
bzr remerge --show-base
```

Re-do the merge of "foobar", using the weave merge algorithm, with additional processing to reduce the size of conflict regions:

```
bzr remerge --merge-type weave --reprocess foobar
```

4.41 remove

Purpose Remove files or directories.

Usage bzr remove [FILE...]

Options

-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
--usage	Show usage message and options.
--new	Only remove files that have never been committed.
-h, --help	Show help message.

Deletion Strategy:

--keep	Delete from bzr but leave the working copy.
--no-backup	Don't backup changed files.
--safe	Backup changed files (default).

Description This makes Bazaar stop tracking changes to the specified files. Bazaar will delete them if they can easily be recovered using revert otherwise they will be backed up (adding an extension of the form .~#~). If no options or parameters are given Bazaar will scan for files that are being tracked by Bazaar but missing in your tree and stop tracking them for you.

Aliases rm, del

4.42 remove-branch

Purpose Remove a branch.

Usage bzr remove-branch [LOCATION]

Options

--force	Remove branch even if it is the active branch.
-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
-h, --help	Show help message.

Description This will remove the branch from the specified location but will keep any working tree or repository in place.

Examples Remove the branch at repo/trunk:

```
bzr remove-branch repo/trunk
```

Aliases rmbranch

4.43 remove-tree

Purpose Remove the working tree from a given branch/checkout.

Usage bzr remove-tree [LOCATION...]

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
--force	Remove the working tree even if it has uncommitted or shelved changes.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description Since a lightweight checkout is little more than a working tree this will refuse to run against one.

To re-create the working tree, use "bzr checkout".

See also *checkout*, *working-trees*

4.44 renames

Purpose Show list of renamed files.

Usage bzr renames [DIR]

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

See also *status*

4.45 resolve

Purpose Mark a conflict as resolved.

Usage bzr resolve [FILE...]

Options

--all	Resolve all conflicts in this tree.
-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
-h, --help	Show help message.

action:

--action=ARG	How to resolve the conflict.
--done	Marks the conflict as resolved.
--take-other	Resolve the conflict taking the merged version into account.
--take-this	Resolve the conflict preserving the version in the working tree.

Description Merge will do its best to combine the changes in two branches, but there are some kinds of problems only a human can fix. When it encounters those, it will mark a conflict. A conflict means that you need to fix something, before you can commit.

Once you have fixed a problem, use "bzr resolve" to automatically mark text conflicts as fixed, "bzr resolve FILE" to mark a specific conflict as resolved, or "bzr resolve –all" to mark all conflicts as resolved.

Aliases resolved

See also *conflicts*

4.46 revert

Purpose Set files in the working tree back to the contents of a previous revision.

Usage bzr revert [FILE...]

Options

-v, --verbose	Display more information.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
--forget-merges	Remove pending merge marker, without changing any files.
--usage	Show usage message and options.
--no-backup	Do not save backups of reverted files.
-r ARG, --revision=ARG	See "help revisionspec" for details.

Description Giving a list of files will revert only those files. Otherwise, all files will be reverted. If the revision is not specified with '–revision', the working tree basis revision is used. A revert operation affects only the working tree, not any revision history like the branch and repository or the working tree basis revision.

To remove only some changes, without reverting to a prior version, use merge instead. For example, "merge . -r -2..-3" (don't forget the ".") will remove the changes introduced by the second last commit (-2), without affecting the changes introduced by the last commit (-1). To remove certain changes on a hunk-by-hunk basis, see the shelve command. To update the branch to a specific revision or the latest revision and update the working tree accordingly while preserving local changes, see the update command.

Uncommitted changes to files that are reverted will be discarded. Howver, by default, any files that have been manually changed will be backed up first. (Files changed only by merge are not backed up.) Backup files have '.~#~' appended to their name, where # is a number.

When you provide files, you can use their current pathname or the pathname from the target revision. So you can use revert to "undelete" a file by name. If you name a directory, all the contents of that directory will be reverted.

If you have newly added files since the target revision, they will be removed. If the files to be removed have been changed, backups will be created as above. Directories containing unknown files will not be deleted.

The working tree contains a list of revisions that have been merged but not yet committed. These revisions will be included as additional parents of the next commit. Normally, using revert clears that list as well as reverting the files. If any files are specified, revert leaves the list of uncommitted merges alone and reverts only the files. Use `bzr revert .` in the tree root to revert all files but keep the recorded merges, and `bzr revert --forget-merges` to clear the pending merge list without reverting any files.

Using "bzr revert –forget-merges", it is possible to apply all of the changes from a branch in a single revision. To do this, perform the merge as desired. Then doing revert with the "–forget-merges" option will keep the content of the tree as it was, but it will clear the list of pending merges. The next commit will then contain all of the changes that are present in the other branch, but without any other parent revisions. Because this technique forgets where these changes originated, it may cause additional conflicts on later merges involving the same source and target branches.

See also *cat*, *export*, *merge*, *shelve*

4.47 revno

Purpose Show current revision number.

Usage bzr revno [LOCATION]

Options

-v, --verbose	Display more information.
-h, --help	Show help message.
--tree	Show revno of working tree.
-q, --quiet	Only display errors and warnings.
--usage	Show usage message and options.
-r ARG, --revision=ARG	See "help revisionspec" for details.

Description This is equal to the number of revisions on this branch.

See also *info*

4.48 root

Purpose Show the tree root directory.

Usage bzr root [FILENAME]

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description The root is the nearest enclosing directory with a .bzr control directory.

4.49 send

Purpose Mail or create a merge-directive for submitting changes.

Usage bzr send [SUBMIT_BRANCH] [PUBLIC_BRANCH]

Options

--body=ARG	Body for the email.
--remember	Remember submit and public branch.
-f ARG, --from=ARG	Branch to generate the submission from, rather than the one containing the working directory.
-v, --verbose	Display more information.
--mail-to=ARG	Mail the request to this address.
--format=ARG	Use the specified output format.

--no-bundle Do not include a bundle in the merge directive.

-q, --quiet Only display errors and warnings.

--strict Refuse to send if there are uncommitted changes in the working tree, –no-strict disables the check.

--usage Show usage message and options.

-o ARG, --output=ARG Write merge directive to this file or directory; use - for stdout.

-m ARG, --message=ARG Message string.

-r ARG, --revision=ARG See "help revisionspec" for details.

--no-patch Do not include a preview patch in the merge directive.

-h, --help Show help message.

Description A merge directive provides many things needed for requesting merges:

- A machine-readable description of the merge to perform
- An optional patch that is a preview of the changes requested
- An optional bundle of revision data, so that the changes can be applied directly from the merge directive, without retrieving data from a branch.

bzr send creates a compact data set that, when applied using bzr merge, has the same effect as merging from the source branch.

By default the merge directive is self-contained and can be applied to any branch containing submit_branch in its ancestory without needing access to the source branch.

If –no-bundle is specified, then Bazaar doesn't send the contents of the revisions, but only a structured request to merge from the public_location. In that case the public_branch is needed and it must be up-to-date and accessible to the recipient. The public_branch is always included if known, so that people can check it later.

The submit branch defaults to the parent of the source branch, but can be overridden. Both submit branch and public branch will be remembered in branch.conf the first time they are used for a particular branch. The source branch defaults to that containing the working directory, but can be changed using –from.

Both the submit branch and the public branch follow the usual behavior with respect to –remember: If there is no default location set, the first send will set it (use –no-remember to avoid setting it). After that, you can omit the location to use the default. To change the default, use –remember. The value will only be saved if the location can be accessed.

In order to calculate those changes, bzr must analyse the submit branch. Therefore it is most efficient for the submit branch to be a local mirror. If a public location is known for the submit_branch, that location is used in the merge directive.

The default behaviour is to send the merge directive by mail, unless -o is given, in which case it is sent to a file.

Mail is sent using your preferred mail program. This should be transparent on Windows (it uses MAPI). On Unix, it requires the xdg-email utility. If the preferred client can't be found (or used), your editor will be used.

To use a specific mail program, set the mail_client configuration option. (For Thunderbird 1.5, this works around some bugs.) Supported values for specific clients are "claws", "evolution", "kmail",

"mail.app" (MacOS X's Mail.app), "mutt", and "thunderbird"; generic options are "default", "editor", "emacsclient", "mapi", and "xdg-email". Plugins may also add supported clients.

If mail is being sent, a to address is required. This can be supplied either on the commandline, by setting the submit_to configuration option in the branch itself or the child_submit_to configuration option in the submit branch.

Two formats are currently supported: "4" uses revision bundle format 4 and merge directive format 2. It is significantly faster and smaller than older formats. It is compatible with Bazaar 0.19 and later. It is the default. "0.9" uses revision bundle format 0.9 and merge directive format 1. It is compatible with Bazaar 0.12 - 0.18.

The merge directives created by bzr send may be applied using bzr merge or bzr pull by specifying a file containing a merge directive as the location.

bzr send makes extensive use of public locations to map local locations into URLs that can be used by other people. See *bzr help configuration* to set them, and use *bzr info* to display them.

See also *merge*, *pull*

4.50 serve

Purpose Run the bzr server.

Usage bzr serve

Options

-v, --verbose Display more information.

-q, --quiet Only display errors and warnings.

-d ARG, --directory=ARG Serve contents of this directory.

--client-timeout=ARG Override the default idle client timeout (5min).

--usage Show usage message and options.

--allow-writes By default the server is a readonly server. Supplying –allow-writes enables write access to the contents of the served directory and below. Note that `bzr serve` does not perform authentication, so unless some form of external authentication is arranged supplying this option leads to global uncontrolled write access to your file system.

--listen=ARG Listen for connections on nominated address.

--port=ARG Listen for connections on nominated port. Passing 0 as the port number will result in a dynamically allocated port. The default port depends on the protocol.

--inet Serve on stdin/out for use from inetd or sshd.

-h, --help Show help message.

protocol:

--protocol=ARG Protocol to serve.

--bzr The Bazaar smart server protocol over TCP. (default port: 4155)

Aliases server

4.51 shelve

Purpose Temporarily set aside some changes from the current tree.

Usage bzr shelve [FILE...]

Options

--all	Shelve all changes.
-v, --verbose	Display more information.
--list	List shelved changes.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
--destroy	Destroy removed changes instead of shelving them.
-m ARG, --message=ARG	Message string.
-r ARG, --revision=ARG	See "help revisionspec" for details.

writer:

--plain	Plaintext diff output.

Description Shelve allows you to temporarily put changes you've made "on the shelf", ie. out of the way, until a later time when you can bring them back from the shelf with the 'unshelve' command. The changes are stored alongside your working tree, and so they aren't propagated along with your branch nor will they survive its deletion.

If shelve –list is specified, previously-shelved changes are listed.

Shelve is intended to help separate several sets of changes that have been inappropriately mingled. If you just want to get rid of all changes and you don't need to restore them later, use revert. If you want to shelve all text changes at once, use shelve –all.

If filenames are specified, only the changes to those files will be shelved. Other files will be left untouched.

If a revision is specified, changes since that revision will be shelved.

You can put multiple items on the shelf, and by default, 'unshelve' will restore the most recently shelved changes.

For complicated changes, it is possible to edit the changes in a separate editor program to decide what the file remaining in the working copy should look like. To do this, add the configuration option

    change_editor = PROGRAM @new_path @old_path

where @new_path is replaced with the path of the new version of the file and @old_path is replaced with the path of the old version of the file. The PROGRAM should save the new file with the desired contents of the file in the working tree.

See also *configuration*, *unshelve*

4.52 sign-my-commits

Purpose Sign all commits by a given committer.

Usage bzr sign-my-commits [LOCATION] [COMMITTER]

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
--dry-run	Don't actually sign anything, just print the revisions that would be signed.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description If location is not specified the local tree is used. If committer is not specified the default committer is used.

This does not sign commits that already have signatures.

4.53 split

Purpose Split a subdirectory of a tree into a separate tree.

Usage bzr split TREE

Options

--usage	Show usage message and options.
-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

Description This command will produce a target tree in a format that supports rich roots, like 'rich-root' or 'rich-root-pack'. These formats cannot be converted into earlier formats like 'dirstate-tags'.

The TREE argument should be a subdirectory of a working tree. That subdirectory will be converted into an independent tree, with its own branch. Commits in the top-level tree will not apply to the new subtree.

See also *join*

4.54 status

Purpose Display status summary.

Usage bzr status [FILE...]

Options

-S, --short	Use short status indicators.
-v, --verbose	Display more information.

-V, --versioned Only show versioned files.

--no-pending Don't show pending merges.

-h, --help Show help message.

-q, --quiet Only display errors and warnings.

--no-classify Do not mark object type using indicator.

--usage Show usage message and options.

--show-ids Show internal object ids.

-c ARG, --change=ARG Select changes introduced by the specified revision. See also "help revisionspec".

-r ARG, --revision=ARG See "help revisionspec" for details.

Description This reports on versioned and unknown files, reporting them grouped by state. Possible states are:

added Versioned in the working copy but not in the previous revision.

removed Versioned in the previous revision but removed or deleted in the working copy.

renamed Path of this file changed from the previous revision; the text may also have changed. This includes files whose parent directory was renamed.

modified Text has changed since the previous revision.

kind changed File kind has been changed (e.g. from file to directory).

unknown Not versioned and not matching an ignore pattern.

Additionally for directories, symlinks and files with a changed executable bit, Bazaar indicates their type using a trailing character: '/', '@' or '*' respectively. These decorations can be disabled using the '–no-classify' option.

To see ignored files use 'bzr ignored'. For details on the changes to file texts, use 'bzr diff'.

Note that –short or -S gives status flags for each item, similar to Subversion's status command. To get output similar to svn -q, use bzr status -SV.

If no arguments are specified, the status of the entire working directory is shown. Otherwise, only the status of the specified files or directories is reported. If a directory is given, status is reported for everything inside that directory.

Before merges are committed, the pending merge tip revisions are shown. To see all pending merge revisions, use the -v option. To skip the display of pending merge information altogether, use the no-pending option or specify a file/directory.

To compare the working directory to a specific revision, pass a single revision to the revision argument.

To see which files have changed in a specific revision, or between two revisions, pass a revision range to the revision argument. This will produce the same results as calling 'bzr diff –summarize'.

Aliases st, stat

See also *diff*, *revert*, *status-flags*

4.55 switch

Purpose Set the branch of a checkout and update.

Usage bzr switch [TO_LOCATION]

Options

--force	Switch even if local commits will be lost.
-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
-r ARG, --revision=ARG	See "help revisionspec" for details.
-b, --create-branch	Create the target branch from this one before switching to it.
--store	Store and restore uncommitted changes in the branch.
-h, --help	Show help message.

Description For lightweight checkouts, this changes the branch being referenced. For heavyweight checkouts, this checks that there are no local commits versus the current bound branch, then it makes the local branch a mirror of the new location and binds to it.

In both cases, the working tree is updated and uncommitted changes are merged. The user can commit or revert these as they desire.

Pending merges need to be committed or reverted before using switch.

The path to the branch to switch to can be specified relative to the parent directory of the current branch. For example, if you are currently in a checkout of /path/to/branch, specifying 'newbranch' will find a branch at /path/to/newbranch.

Bound branches use the nickname of its master branch unless it is set locally, in which case switching will update the local nickname to be that of the master.

4.56 tag

Purpose Create, remove or modify a tag naming a revision.

Usage bzr tag [TAG_NAME]

Options

--force	Replace existing tags.
-v, --verbose	Display more information.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch in which to place the tag.
--usage	Show usage message and options.
-r ARG, --revision=ARG	See "help revisionspec" for details.
--delete	Delete this tag rather than placing it.

Description Tags give human-meaningful names to revisions. Commands that take a -r (–revision) option can be given -rtag:X, where X is any previously created tag.

Tags are stored in the branch. Tags are copied from one branch to another along when you branch, push, pull or merge.

It is an error to give a tag name that already exists unless you pass –force, in which case the tag is moved to point to the new revision.

To rename a tag (change the name but keep it on the same revsion), run `bzr tag new-name -r tag:old-name` and then `bzr tag --delete oldname`.

If no tag name is specified it will be determined through the 'automatic_tag_name' hook. This can e.g. be used to automatically tag upstream releases by reading configure.ac. See `bzr help hooks` for details.

See also *commit*, *tags*

4.57 tags

Purpose List tags.

Usage bzr tags

Options

--sort=ARG	Sort tags by different criteria.
-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch whose tags should be displayed.
--usage	Show usage message and options.
--show-ids	Show internal object ids.
-r ARG, --revision=ARG	See "help revisionspec" for details.
-h, --help	Show help message.

Description This command shows a table of tag names and the revisions they reference.

See also *tag*

4.58 testament

Purpose Show testament (signing-form) of a revision.

Usage bzr testament [BRANCH]

Options

-v, --verbose	Display more information.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
--long	Produce long-format testament.
--strict	Produce a strict-format testament.

--usage Show usage message and options.

-r ARG, --revision=ARG See "help revisionspec" for details.

4.59 unbind

Purpose Convert the current checkout into a regular branch.

Usage bzr unbind

Options

--usage Show usage message and options.

-d ARG, --directory=ARG Branch to operate on, instead of working directory.

-q, --quiet Only display errors and warnings.

-v, --verbose Display more information.

-h, --help Show help message.

Description After unbinding, the local branch is considered independent and subsequent commits will be local only.

See also *bind*, *checkouts*

4.60 uncommit

Purpose Remove the last committed revision.

Usage bzr uncommit [LOCATION]

Options

--dry-run Don't actually make changes.

-v, --verbose Display more information.

--keep-tags Keep tags that point to removed revisions.

-h, --help Show help message.

-q, --quiet Only display errors and warnings.

--usage Show usage message and options.

--force Say yes to all questions.

--local Only remove the commits from the local branch when in a checkout.

-r ARG, --revision=ARG See "help revisionspec" for details.

Description –verbose will print out what is being removed. –dry-run will go through all the motions, but not actually remove anything.

If –revision is specified, uncommit revisions to leave the branch at the specified revision. For example, "bzr uncommit -r 15" will leave the branch at revision 15.

Uncommit leaves the working tree ready for a new commit. The only change it may make is to restore any pending merges that were present before the commit.

See also *commit*

4.61 unshelve

Purpose Restore shelved changes.

Usage bzr unshelve [SHELF_ID]

Options

-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--usage	Show usage message and options.
-h, --help	Show help message.

action:

--apply	Apply changes and remove from the shelf.
--delete-only	Delete changes without applying them.
--dry-run	Show changes, but do not apply or remove them.
--keep	Apply changes but don't delete them.
--preview	Instead of unshelving the changes, show the diff that would result from unshelving.

Description By default, the most recently shelved changes are restored. However if you specify a shelf by id those changes will be restored instead. This works best when the changes don't depend on each other.

See also *shelve*

4.62 update

Purpose Update a working tree to a new revision.

Usage bzr update [DIR]

Options

-v, --verbose	Display more information.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
--usage	Show usage message and options.
--show-base	Show base revision text in conflicts.
-r ARG, --revision=ARG	See "help revisionspec" for details.

Description This will perform a merge of the destination revision (the tip of the branch, or the specified revision) into the working tree, and then make that revision the basis revision for the working tree.

You can use this to visit an older revision, or to update a working tree that is out of date from its branch.

If there are any uncommitted changes in the tree, they will be carried across and remain as uncommitted changes after the update. To discard these changes, use 'bzr revert'. The uncommitted changes may conflict with the changes brought in by the change in basis revision.

If the tree's branch is bound to a master branch, bzr will also update the branch from the master.

You cannot update just a single file or directory, because each Bazaar working tree has just a single basis revision. If you want to restore a file that has been removed locally, use 'bzr revert' instead of 'bzr update'. If you want to restore a file to its state in a previous revision, use 'bzr revert' with a '-r' option, or use 'bzr cat' to write out the old content of that file to a new location.

The 'dir' argument, if given, must be the location of the root of a working tree to update. By default, the working tree that contains the current working directory is used.

Aliases up

See also *pull*, *status-flags*, *working-trees*

4.63 upgrade

Purpose Upgrade a repository, branch or working tree to a newer format.

Usage bzr upgrade [URL]

Options

--dry-run	Show what would be done, but don't actually do anything.
-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
--clean	Remove the backup.bzr directory if successful.
--usage	Show usage message and options.
-h, --help	Show help message.

Branch format:

--format=ARG	Upgrade to a specific format. See "bzr help formats" for details.
--2a	Format for the bzr 2.0 series. Uses group-compress storage. Provides rich roots which are a one-way transition.
--default	Format for the bzr 2.0 series. Uses group-compress storage. Provides rich roots which are a one-way transition.
--development-colo	The 2a format with experimental support for colocated branches.

–pack-0.92 Pack-based format used in 1.x series. Introduced in 0.92. Interoperates with bzr repositories before 0.92 but cannot be read by bzr < 0.92.

Description When the default format has changed after a major new release of Bazaar, you may be informed during certain operations that you should upgrade. Upgrading to a newer format may improve performance or make new features available. It may however limit interoperability with older repositories or with older versions of Bazaar.

If you wish to upgrade to a particular format rather than the current default, that can be specified using the –format option. As a consequence, you can use the upgrade command this way to "downgrade" to an earlier format, though some conversions are a one way process (e.g. changing from the 1.x default to the 2.x default) so downgrading is not always possible.

A backup.bzr.~#~ directory is created at the start of the conversion process (where # is a number). By default, this is left there on completion. If the conversion fails, delete the new .bzr directory and rename this one back in its place. Use the –clean option to ask for the backup.bzr directory to be removed on successful conversion. Alternatively, you can delete it by hand if everything looks good afterwards.

If the location given is a shared repository, dependent branches are also converted provided the repository converts successfully. If the conversion of a branch fails, remaining branches are still tried.

For more information on upgrades, see the Bazaar Upgrade Guide, http://doc.bazaar.canonical.com/latest/en/upgrade-guide/.

See also *check*, *formats*, *reconcile*

4.64 verify-signatures

Purpose Verify all commit signatures.

Usage bzr verify-signatures [LOCATION]

Options

-v, --verbose	Display more information.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
-k ARG, --acceptable-keys=ARG	Comma separated list of GPG key patterns which are acceptable for verification.
--usage	Show usage message and options.
-r ARG, --revision=ARG	See "help revisionspec" for details.

Description Verifies that all commits in the branch are signed by known GnuPG keys.

4.65 version

Purpose Show version of bzr.

Usage bzr version

Options

--usage	Show usage message and options.
--short	Print just the version number.

-q, --quiet	Only display errors and warnings.
-v, --verbose	Display more information.
-h, --help	Show help message.

4.66 version-info

Purpose Show version information about this tree.

Usage bzr version-info [LOCATION]

Options

--all	Include all possible information.
-v, --verbose	Display more information.
--include-history	Include the revision-history.
--check-clean	Check if tree is clean.
-q, --quiet	Only display errors and warnings.
--include-file-revisions	Include the last revision for each file.
--template=ARG	Template for the output.
--usage	Show usage message and options.
-r ARG, --revision=ARG	See "help revisionspec" for details.
-h, --help	Show help message.

format:

--format=ARG	Select the output format.
--custom	Version info in Custom template-based format.
--python	Version info in Python format.
--rio	Version info in RIO (simple text) format (default).

Description You can use this command to add information about version into source code of an application. The output can be in one of the supported formats or in a custom format based on a template.

For example:

```
bzr version-info --custom \
  --template="#define VERSION_INFO \"Project 1.2.3 (r{revno})\"\n"
```

will produce a C header file with formatted string containing the current revision number. Other supported variables in templates are:

- {date} - date of the last revision
- {build_date} - current date
- {revno} - revision number
- {revision_id} - revision id
- {branch_nick} - branch nickname
- **{clean} - 0 if the source tree contains uncommitted changes,** otherwise 1

4.67 view

Purpose Manage filtered views.

Usage bzr view [FILE...]

Options

--all	Apply list or delete action to all views.
-v, --verbose	Display more information.
-h, --help	Show help message.
-q, --quiet	Only display errors and warnings.
--name=ARG	Name of the view to define, list or delete.
--switch=ARG	Name of the view to switch to.
--usage	Show usage message and options.
--delete	Delete the view.

Description Views provide a mask over the tree so that users can focus on a subset of a tree when doing their work. After creating a view, commands that support a list of files - status, diff, commit, etc - effectively have that list of files implicitly given each time. An explicit list of files can still be given but those files must be within the current view.

In most cases, a view has a short life-span: it is created to make a selected change and is deleted once that change is committed. At other times, you may wish to create one or more named views and switch between them.

To disable the current view without deleting it, you can switch to the pseudo view called `off`. This can be useful when you need to see the whole tree for an operation or two (e.g. merge) but want to switch back to your view after that.

Examples To define the current view:

```
bzr view file1 dir1 ...
```

To list the current view:

```
bzr view
```

To delete the current view:

```
bzr view --delete
```

To disable the current view without deleting it:

```
bzr view --switch off
```

To define a named view and switch to it:

```
bzr view --name view-name file1 dir1 ...
```

To list a named view:

```
bzr view --name view-name
```

To delete a named view:

```
bzr view --name view-name --delete
```

To switch to a named view:

```
bzr view --switch view-name
```

To list all views defined:

```
bzr view --all
```

To delete all views:

```
bzr view --delete --all
```

4.68 whoami

Purpose Show or set bzr user id.

Usage bzr whoami [NAME]

Options

-v, --verbose	Display more information.
-q, --quiet	Only display errors and warnings.
-d ARG, --directory=ARG	Branch to operate on, instead of working directory.
--branch	Set identity for the current branch instead of globally.
--usage	Show usage message and options.
--email	Display email address only.
-h, --help	Show help message.

Examples Show the email of the current user:

```
bzr whoami --email
```

Set the current user:

```
bzr whoami "Frank Chu <fchu@example.com>"
```