Kallithea Documentation

Release 0.7.99

Kallithea Developers

Mar 31, 2023

Contents

- genindex
- search

CHAPTER 1

Readme

1.1 Kallithea README

1.1.1 About

Kallithea is a fast and powerful management tool for Mercurial and Git with a built-in push/pull server, full text search and code-review. It works on HTTP/HTTPS and SSH, has a built-in permission/authentication system with the ability to authenticate via LDAP or ActiveDirectory. Kallithea also provides simple API so it's easy to integrate with existing external systems.

Kallithea is similar in some respects to GitHub or Bitbucket, however Kallithea can be run as standalone hosted application on your own server. It is open-source and focuses more on providing a customised, self-administered interface for Mercurial and Git repositories. Kallithea works on Unix-like systems and Windows.

Kallithea was forked from RhodeCode in July 2014 and has been heavily modified.

1.1.2 Installation

Kallithea requires Python 3 and it is recommended to install it in a virtualenv. Official releases of Kallithea can be installed with:

```
pip install kallithea
```

The development repository is kept very stable and used in production by the developers – you can do the same.

Please visit https://docs.kallithea-scm.org/en/latest/installation.html for more details.

There is also an experimental Puppet module for installing and setting up Kallithea. Currently, only basic functionality is provided, but it is still enough to get up and running quickly, especially for people without Python background. See https://docs.kallithea-scm.org/en/latest/installation_puppet.html for further information.

1.1.3 Source code

The latest sources can be obtained from https://kallithea-scm.org/repos/kallithea.

1.1.4 Kallithea features

- Has its own middleware to handle Mercurial and Git protocol requests. Each request is authenticated and logged together with IP address.
- Built for speed and performance. You can make multiple pulls/pushes simultaneously. Proven to work with thousands of repositories and users.
- Supports HTTP/HTTPS with LDAP, AD, or proxy-pass authentication.
- Supports SSH access with server-side public key management.
- Full permissions (private/read/write/admin) together with IP restrictions for each repository, additional explicit forking, repositories group and repository creation permissions.
- User groups for easier permission management.
- Repository groups let you group repos and manage them easier. They come with permission delegation features, so you can delegate groups management.
- Users can fork other users repos, and compare them at any time.
- Built-in versioned paste functionality (Gist) for sharing code snippets.
- Integrates easily with other systems, with custom created mappers you can connect it to almost any issue tracker, and with a JSON-RPC API you can make much more.
- Built-in commit API lets you add, edit and commit files right from Kallithea web interface using simple editor or upload binary files using simple form.
- Powerful pull request driven review system with inline commenting, changeset statuses, and notification system.
- Importing and syncing repositories from remote locations for Git and Mercurial.
- Mako templates let you customize the look and feel of the application.
- Beautiful diffs, annotations and source code browsing all colored by pygments. Raw diffs are made in Git-diff format for both VCS systems, including Git binary-patches.
- Mercurial and Git DAG graphs and Flot-powered graphs with zooming and statistics to track activity for repositories.
- Admin interface with user/permission management. Admin activity journal logs pulls, pushes, forks, registrations and other actions made by all users.
- Server side forks. It is possible to fork a project and modify it freely without breaking the main repository.
- reST and Markdown README support for repositories.
- Full text search powered by Whoosh on the source files, commit messages, and file names. Built-in indexing daemons, with optional incremental index build (no external search servers required all in one application).
- Setup project descriptions/tags and info inside built in DB for easy, non-filesystem operations.
- Intelligent cache with invalidation after push or project change, provides high performance and always up to date data.
- RSS/Atom feeds, Gravatar support, downloadable sources as zip/tar/gz.
- Optional async tasks for speed and performance using Celery.

- Backup scripts can do backup of whole app and send it over scp to desired location.
- Based on TurboGears2, SQLAlchemy, Whoosh, Bootstrap, and other open source libraries.
- Uses PostgreSQL, SQLite, or MariaDB/MySQL databases.

1.1.5 License

Kallithea is released under the GPLv3 license. Kallithea is a Software Freedom Conservancy project and thus controlled by a non-profit organization. No commercial entity can take ownership of the project and change the direction.

Kallithea started out as an effort to make sure the existing GPLv3 codebase would stay available under a legal license. Kallithea thus has to stay GPLv3 compatible ... but we are also happy it is GPLv3 and happy to keep it that way. A different license (such as AGPL) could perhaps help attract a different community with a different mix of Free Software people and companies but we are happy with the current focus.

1.1.6 Community

Kallithea is maintained by its users who contribute the fixes they would like to see.

Get in touch with the rest of the community:

- Join the mailing list users and developers – see http://lists.sfconservancy.org/mailman/listinfo/kallithea-general.
- Use IRC and join #kallithea on FreeNode (irc.freenode.net) or use http://webchat.freenode.net/?channels=kallithea.
- Follow Kallithea on Twitter, **@KallitheaSCM**.
- Please report issues on the mailing list. An archive of the old issue tracker is available at: https://kallithea-scm.org/bitbucket-archive/issues/index.html

> **Note:** Please try to read the documentation before posting any issues, especially the **troubleshooting section**

1.1.7 Online documentation

Online documentation for the current version of Kallithea is available at https://docs.kallithea-scm.org/en/stable/. Documentation for the current development version can be found on https://docs.kallithea-scm.org/en/default/.

You can also build the documentation locally: go to `docs/` and run:

```
make html
```

Note: You need to have Sphinx installed to build the documentation. If you don't have Sphinx installed you can install it via the command: `pip install sphinx`.

1.1.8 Migrating from RhodeCode

Kallithea 0.3.2 and earlier supports migrating from an existing RhodeCode installation. To migrate, install Kallithea 0.3.2 and follow the instructions in the 0.3.2 README to perform a one-time conversion of the database from RhodeCode to Kallithea, before upgrading to this version of Kallithea.

CHAPTER 2

Administrator guide

Installation and upgrade

2.1 Installation overview

Some overview and some details that can help understanding the options when installing Kallithea.

1. **Prepare environment and external dependencies.** Kallithea needs:
    - A filesystem where the Mercurial and Git repositories can be stored.
    - A database where meta data can be stored.
    - A Python environment where the Kallithea application and its dependencies can be installed.
    - A web server that can host the Kallithea web application using the WSGI API.
2. **Install Kallithea software.** This makes the `kallithea-cli` command line tool available.
3. **Prepare front-end files** Some front-end files must be fetched or created using `npm` and `node` tooling so they can be served to the client as static files.
4. **Create low level configuration file.** Use `kallithea-cli config-create` to create a `.ini` file with database connection info, mail server information, configuration for the specified web server, etc.
5. **Populate the database.** Use `kallithea-cli db-create` with the `.ini` file to create the database schema and insert the most basic information: the location of the repository store and an initial local admin user.
6. **Configure the web server.** The web server must invoke the WSGI entrypoint for the Kallithea software using the `.ini` file (and thus the database). This makes the web application available so the local admin user can log in and tweak the configuration further.
7. **Configure users.** The initial admin user can create additional local users, or configure how users can be created and authenticated from other user directories.

See the subsequent sections, the separate OS-specific instructions, and *Setup* for details on these steps.

2.1.1 File system location

Kallithea can be installed in many different ways. The main parts are:

- A location for the Kallithea software and its dependencies. This includes the Python code, template files, and front-end code. After installation, this will be read-only (except when upgrading).
- A location for the `.ini` configuration file that tells the Kallithea instance which database to use (and thus also the repository location). After installation, this will be read-only (except when upgrading).
- A location for various data files and caches for the Kallithea instance. This is by default in a `data` directory next to the `.ini` file. This will have to be writable by the running Kallithea service.
- A database. The `.ini` file specifies which database to use. The database will be a separate service and live elsewhere in the filesystem if using PostgreSQL or MariaDB/MySQL. If using SQLite, it will by default live next to the `.ini` file, as `kallithea.db`.
- A location for the repositories that are hosted by this Kallithea instance. This will have to be writable by the running Kallithea service. The path to this location will be configured in the database.

For production setups, one recommendation is to use `/srv/kallithea` for the `.ini` and `data`, place the virtualenv in `venv`, and use a Kallithea clone in `kallithea`. Create a `kallithea` user, let it own `/srv/kallithea`, and run as that user when installing.

For simple setups, it is fine to just use something like a `kallithea` user with home in `/home/kallithea` and place everything there.

For experiments, it might be convenient to run everything as yourself and work inside a clone of Kallithea, with the `.ini` and SQLite database in the root of the clone, and a virtualenv in `venv`.

2.1.2 Python environment

Kallithea is written entirely in Python and requires Python version 3.6 or higher.

Given a Python installation, there are different ways of providing the environment for running Python applications. Each of them pretty much corresponds to a `site-packages` directory somewhere where packages can be installed.

Kallithea itself can be run from source or be installed, but even when running from source, there are some dependencies that must be installed in the Python environment used for running Kallithea.

- Packages *could* be installed in Python's `site-packages` directory ... but that would require running pip as root and it would be hard to uninstall or upgrade and is probably not a good idea unless using a package manager.
- Packages could also be installed in `~/.local` ... but that is probably only a good idea if using a dedicated user per application or instance.
- Finally, it can be installed in a virtualenv. That is a very lightweight "container" where each Kallithea instance can get its own dedicated and self-contained virtual environment.

We recommend using virtualenv for installing Kallithea.

2.1.3 Locale environment

In order to ensure a correct functioning of Kallithea with respect to non-ASCII characters in user names, file paths, commit messages, etc., it is very important that Kallithea is run with a correct *locale* configuration.

On Unix, environment variables like `LANG` or `LC_ALL` can specify a language (like `en_US`) and encoding (like `UTF-8`) to use for code points outside the ASCII range. The flexibility of supporting multiple encodings of Unicode has the flip side of having to specify which encoding to use - especially for Mercurial.

It depends on the OS distribution and system configuration which locales are available. For example, some Docker containers based on Debian default to only supporting the `C` language, while other Linux environments have `en_US` but not `C`. The `locale -a` command will show which values are available on the current system. Regardless of the actual language, you should normally choose a locale that has the `UTF-8` encoding (note that spellings `utf8`, `utf-8`, `UTF8`, `UTF-8` are all referring to the same thing)

For technical reasons, the locale configuration **must** be provided in the environment in which Kallithea runs - it cannot be specified in the `.ini` file. How to practically do this depends on the web server that is used and the way it is started. For example, gearbox is often started by a normal user, either manually or via a script. In this case, the required locale environment variables can be provided directly in that user's environment or in the script. However, web servers like Apache are often started at boot via an init script or service file. Modifying the environment for this case would thus require root/administrator privileges. Moreover, that environment would dictate the settings for all web services running under that web server, Kallithea being just one of them. Specifically in the case of Apache with `mod_wsgi`, the locale can be set for a specific service in its `WSGIDaemonProcess` directive, using the `lang` parameter.

2.1.4 Installation methods

Kallithea must be installed on a server. Kallithea is installed in a Python environment so it can use packages that are installed there and make itself available for other packages.

Two different cases will pretty much cover the options for how it can be installed.

- The Kallithea source repository can be cloned and used – it is kept stable and can be used in production. The Kallithea maintainers use the development branch in production. The advantage of installation from source and regularly updating it is that you take advantage of the most recent improvements. Using it directly from a DVCS also means that it is easy to track local customizations.

  Running `pip install -e .` in the source will use pip to install the necessary dependencies in the Python environment and create a `.../site-packages/Kallithea.egg-link` file there that points at the Kallithea source.

- Kallithea can also be installed from ready-made packages using a package manager. The official released versions are available on PyPI and can be downloaded and installed with all dependencies using `pip install kallithea`.

  With this method, Kallithea is installed in the Python environment as any other package, usually as a `.../site-packages/Kallithea-X-py3.8.egg/` directory with Python files and everything else that is needed.

  (`pip install kallithea` from a source tree will do pretty much the same but build the Kallithea package itself locally instead of downloading it.)

Note: Kallithea includes front-end code that needs to be processed to prepare static files that can be served at run time and used on the client side. The tool npm is used to download external dependencies and orchestrate the processing. The `npm` binary must thus be available at install time but is not used at run time.

2.1.5 Web server

Kallithea is (primarily) a WSGI application that must be run from a web server that serves WSGI applications over HTTP.

Kallithea itself is not serving HTTP (or HTTPS); that is the web server's responsibility. Kallithea does however need to know its own user facing URL (protocol, address, port and path) for each HTTP request. Kallithea will usually use its own HTML/cookie based authentication but can also be configured to use web server authentication.

There are several web server options:

- Kallithea uses the Gearbox tool as command line interface. Gearbox provides `gearbox serve` as a convenient way to launch a Python WSGI / web server from the command line. That is perfect for development and evaluation. Actual use in production might have different requirements and need extra work to make it manageable as a scalable system service.

  Gearbox comes with its own built-in web server for development but Kallithea defaults to using Waitress. Gunicorn and Gevent are also options. These web servers have different limited feature sets.

  The web server used by `gearbox serve` is configured in the `.ini` file. Create it with `config-create` using for example `http_server=waitress` to get a configuration starting point for your choice of web server.

  (Gearbox will do like `paste` and use the WSGI application entry point `kallithea.config.application:make_app` as specified in `setup.py`.)

- Apache httpd can serve WSGI applications directly using mod_wsgi and a simple Python file with the necessary configuration. This is a good option if Apache is an option.
- uWSGI is also a full web server with built-in WSGI module. Use `config-create` with `http_server=uwsgi` to get a `.ini` file with uWSGI configuration.
- IIS can also server WSGI applications directly using isapi-wsgi.
- A reverse HTTP proxy can be put in front of another web server which has WSGI support. Such a layered setup can be complex but might in some cases be the right option, for example to standardize on one internet-facing web server, to add encryption or special authentication or for other security reasons, to provide caching of static files, or to provide load balancing or fail-over. Nginx, Varnish and HAProxy are often used for this purpose, often in front of a `gearbox serve` that somehow is wrapped as a service.

The best option depends on what you are familiar with and the requirements for performance and stability. Also, keep in mind that Kallithea mainly is serving dynamically generated pages from a relatively slow Python process. Kallithea is also often used inside organizations with a limited amount of users and thus no continuous hammering from the internet.

Note: Kallithea, the libraries it uses, and Python itself do in several places use simple caching in memory. Caches and memory are not always released in a way that is suitable for long-running processes. They might appear to be leaking memory. The worker processes should thus regularly be restarted - for example after 1000 requests and/or one hour. This can usually be done by the web server or the tool used for running it as a system service.

2.2 Installation on Unix/Linux

The following describes three different ways of installing Kallithea:

- *Installation from repository source*: The simplest way to keep the installation up-to-date and track any local customizations is to run directly from source in a Kallithea repository clone, preferably inside a virtualenv virtual Python environment.
- *Installing a released version in a virtualenv*: If you prefer to only use released versions of Kallithea, the recommended method is to install Kallithea in a virtual Python environment using *virtualenv*. The advantages of this method over direct installation is that Kallithea and its dependencies are completely contained inside the virtualenv (which also means you can have multiple installations side by side or remove it entirely by just removing the virtualenv directory) and does not require root privileges.
- Kallithea can also be installed with plain pip - globally or with `--user` or similar. The package will be installed in the same location as all other Python packages you have ever installed. As a result, removing it is

not as straightforward as with a virtualenv, as you'd have to remove its dependencies manually and make sure that they are not needed by other packages. We recommend using virtualenv.

Regardless of the installation method you may need to make sure you have appropriate development packages installed, as installation of some of the Kallithea dependencies requires a working C compiler and libffi library headers. Depending on your configuration, you may also need to install Git and development packages for the database of your choice.

For Debian and Ubuntu, the following command will ensure that a reasonable set of dependencies is installed:

```
sudo apt-get install build-essential git libffi-dev python3-dev
```

For Fedora and RHEL-derivatives, the following command will ensure that a reasonable set of dependencies is installed:

```
sudo yum install gcc git libffi-devel python3-devel
```

2.2.1 Installation from repository source

To install Kallithea in a virtualenv using the stable branch of the development repository, use the following commands in your bash shell:

```
hg clone https://kallithea-scm.org/repos/kallithea -u stable
cd kallithea
python3 -m venv venv
. venv/bin/activate
pip install --upgrade pip setuptools
pip install --upgrade -e .
python3 setup.py compile_catalog   # for translation of the UI
```

Note: This will install all Python dependencies into the virtualenv. Kallithea itself will however only be installed as a pointer to the source location. The source clone must thus be kept in the same location, and it shouldn't be updated to other revisions unless you want to upgrade. Edits in the source tree will have immediate impact (possibly after a restart of the service).

You can now proceed to *Prepare front-end files*.

2.2.2 Installing a released version in a virtualenv

It is highly recommended to use a separate virtualenv for installing Kallithea. This way, all libraries required by Kallithea will be installed separately from your main Python installation and other applications and things will be less problematic when upgrading the system or Kallithea. An additional benefit of virtualenv is that it doesn't require root privileges.

- Don't install as root - install as a dedicated user like `kallithea`. If necessary, create the top directory for the virtualenv (like `/srv/kallithea/venv`) as root and assign ownership to the user.

  Make a parent folder for the virtualenv (and perhaps also Kallithea configuration and data files) such as `/srv/kallithea`. Create the directory as root if necessary and grant ownership to the `kallithea` user.

- Create a new virtual environment, for example in `/srv/kallithea/venv`, specifying the right Python binary:

```
python3 -m venv /srv/kallithea/venv
```

- Activate the virtualenv in your current shell session and make sure the basic requirements are up-to-date by running the following commands in your bash shell:

```
. /srv/kallithea/venv/bin/activate
pip install --upgrade pip setuptools
```

Note: You can't use UNIX `sudo` to source the `activate` script; it will "activate" a shell that terminates immediately.

- Install Kallithea in the activated virtualenv:

```
pip install --upgrade kallithea
```

Note: Some dependencies are optional. If you need them, install them in the virtualenv too:

```
pip install --upgrade kallithea python-ldap python-pam psycopg2
```

This might require installation of development packages using your distribution's package manager.

Alternatively, download a .tar.gz from http://pypi.python.org/pypi/Kallithea, extract it and install from source by running:

```
pip install --upgrade .
```

- This will install Kallithea together with all other required Python libraries into the activated virtualenv.

You can now proceed to *Prepare front-end files*.

2.2.3 Prepare front-end files

Finally, the front-end files with CSS and JavaScript must be prepared. This depends on having some commands available in the shell search path: `npm` version 6 or later, and `node.js` (version 12 or later) available as `node`. The installation method for these dependencies varies between operating systems and distributions.

Prepare the front-end by running:

```
kallithea-cli front-end-build
```

You can now proceed to *Setup*.

Warning: This section is outdated and needs updating for Python 3.

2.3 Installation on Windows (7/Server 2008 R2 and newer)

2.3.1 First time install

Target OS: Windows 7 and newer or Windows Server 2008 R2 and newer

Tested on Windows 8.1, Windows Server 2008 R2 and Windows Server 2012

To install on an older version of Windows, see installation_win_old.html

Step 1 – Install Python

Install Python 3. Latest version is recommended. If you need another version, they can run side by side.

- Download Python 3 from http://www.python.org/download/
- Choose and click on the version
- Click on "Windows X86-64 Installer" for x64 or "Windows x86 MSI installer" for Win32.
- Disable UAC or run the installer with admin privileges. If you chose to disable UAC, do not forget to reboot afterwards.

While writing this guide, the latest version was v3.8.1. Remember the specific major and minor versions installed, because they will be needed in the next step. In this case, it is "3.8".

Step 2 – Python BIN

Add Python BIN folder to the path. This can be done manually (editing "PATH" environment variable) or by using Windows Support Tools that come pre-installed in Windows Vista/7 and later.

Open a CMD and type:

```
SETX PATH "%PATH%;[your-python-path]" /M
```

Please substitute [your-python-path] with your Python installation path. Typically this is `C:\\Python38`.

Step 3 – Install pywin32 extensions

Download pywin32 from: http://sourceforge.net/projects/pywin32/files/

- Click on "pywin32" folder
- Click on the first folder (in this case, Build 219, maybe newer when you try)
- Choose the file ending with ".amd64-py3.x.exe" (".win32-py3.x.exe" for Win32) where x is the minor version of Python you installed. When writing this guide, the file was: http://sourceforge.net/projects/pywin32/files/pywin32/Build%20219/pywin32-219.win-amd64-py3.8.exe/download (x64) http://sourceforge.net/projects/pywin32/files/pywin32/Build%20219/pywin32-219.win32-py3.8.exe/download (Win32)

Step 5 – Kallithea folder structure

Create a Kallithea folder structure.

This is only an example to install Kallithea. Of course, you can change it. However, this guide will follow the proposed structure, so please later adapt the paths if you change them. Folders without spaces are recommended.

Create the following folder structure:

```
C:\Kallithea
C:\Kallithea\Bin
C:\Kallithea\Env
C:\Kallithea\Repos
```

Step 6 – Install virtualenv

Note: A python virtual environment will allow for isolation between the Python packages of your system and those used for Kallithea. It is strongly recommended to use it to ensure that Kallithea does not change a dependency that other software uses or vice versa.

To create a virtual environment, run:

```
python3 -m venv C:\Kallithea\Env
```

Step 7 – Install Kallithea

In order to install Kallithea, you need to be able to run "pip install kallithea". It will use pip to install the Kallithea Python package and its dependencies. Some Python packages use managed code and need to be compiled. This can be done on Linux without any special steps. On Windows, you will need to install Microsoft Visual C++ compiler for Python 3.8.

Download and install "Microsoft Visual C++ Compiler for Python 3.8" from http://aka.ms/vcpython27

Note: You can also install the dependencies using already compiled Windows binaries packages. A good source of compiled Python packages is http://www.lfd.uci.edu/~gohlke/pythonlibs/. However, not all of the necessary packages for Kallithea are on this site and some are hard to find, so we will stick with using the compiler.

In a command prompt type (adapting paths if necessary):

```
cd C:\Kallithea\Env\Scripts
activate
pip install --upgrade pip setuptools
```

The prompt will change into "(Env) C:\Kallithea\Env\Scripts" or similar (depending of your folder structure). Then type:

```
pip install kallithea
```

Note: This will take some time. Please wait patiently until it is fully complete. Some warnings will appear. Don't worry, they are normal.

Step 8 – Install Git (optional)

Mercurial being a python package, was installed automatically when doing `pip install kallithea`.

You need to install Git manually if you want Kallithea to be able to host Git repositories. See http://git-scm.com/book/en/v2/Getting-Started-Installing-Git#Installing-on-Windows for instructions. The location of the Git binaries (like `c:\path\to\git\bin`) must be added to the `PATH` environment variable so `git.exe` and other tools like `gzip.exe` are available.

Step 9 – Configuring Kallithea

Steps taken from setup.html

You have to use the same command prompt as in Step 7, so if you closed it, reopen it following the same commands (including the "activate" one). When ready, type:

```
cd C:\Kallithea\Bin
kallithea-cli config-create my.ini
```

Then you must edit my.ini to fit your needs (IP address, IP port, mail settings, database, etc.). NotePad++ or a similar text editor is recommended to properly handle the newline character differences between Unix and Windows.

For the sake of simplicity, run it with the default settings. After your edits (if any) in the previous command prompt, type:

```
kallithea-cli db-create -c my.ini
```

Warning: This time a *new* database will be installed. You must follow a different process to later *upgrade* to a newer Kallithea version.

The script will ask you for confirmation about creating a new database, answer yes (y)

The script will ask you for the repository path, answer C:\Kallithea\Repos (or similar).

The script will ask you for the admin username and password, answer "admin" + "123456" (or whatever you want)

The script will ask you for admin mail, answer "admin@xxxx.com" (or whatever you want).

If you make a mistake and the script doesn't end, don't worry: start it again.

If you decided not to install Git, you will get errors about it that you can ignore.

Step 10 – Running Kallithea

In the previous command prompt, being in the C:\Kallithea\Bin folder, type:

```
gearbox serve -c my.ini
```

Open your web server, and go to http://127.0.0.1:5000

It works!! :-)

Remark: If it does not work the first time, Ctrl-C the CMD process and start it again. Don't forget the "http://" in Internet Explorer.

What this guide does not cover:

- Installing Celery
- Running Kallithea as a Windows Service. You can investigate here:
  - http://pypi.python.org/pypi/wsgisvc
  - http://ryrobes.com/python/running-python-scripts-as-a-windows-service/
  - http://wiki.pylonshq.com/display/pylonscookbook/How+to+run+Pylons+as+a+Windows+service
- Using Apache. You can investigate here:
  - https://groups.google.com/group/rhodecode/msg/c433074e813ffdc4

Warning: This section is outdated and needs updating for Python 3.

2.4 Installation on Windows (XP/Vista/Server 2003/Server 2008)

2.4.1 First-time install

Target OS: Windows XP SP3 32-bit English (Clean installation) + All Windows Updates until 24-may-2012

Note: This installation is for 32-bit systems, for 64-bit Windows you might need to download proper 64-bit versions of the different packages (Windows Installer, Win32py extensions) plus some extra tweaks. These extra steps haven been marked as "64-bit". Tested on Windows Server 2008 R2 SP1, 9-feb-2013. If you run into any 64-bit related problems, please check these pages:

- http://blog.victorjabur.com/2011/06/05/compiling-python-2-7-modules-on-windows-32-and-64-using-msvc-2008-express/
- http://bugs.python.org/issue7511

Step 1 – Install Visual Studio 2008 Express

Optional: You can also install MinGW, but VS2008 installation is easier.

Download "Visual C++ 2008 Express Edition with SP1" from: http://download.microsoft.com/download/E/8/E/E8EEB394-7F42-4963-A2D8-29559B738298/VS2008ExpressWithSP1ENUX1504728.iso (if not found or relocated, google for "visual studio 2008 express" for updated link. This link was taken from http://stackoverflow.com/questions/15318560/visual-c-2008-express-download-link-dead)

You can also download full ISO file for offline installation, just choose "All – Offline Install ISO image file" in the previous page and choose "Visual C++ 2008 Express" when installing.

Note: Using other versions of Visual Studio will lead to random crashes. You must use Visual Studio 2008!"

Note: Silverlight Runtime and SQL Server 2008 Express Edition are not required, you can uncheck them

Note: 64-bit: You also need to install the Microsoft Windows SDK for .NET 3.5 SP1 (.NET 4.0 won't work). Download from: http://www.microsoft.com/en-us/download/details.aspx?id=3138

Note: 64-bit: You also need to copy and rename a .bat file to make the Visual C++ compiler work. I am not sure why this is not necessary for 32-bit. Copy C:Program Files (x86)Microsoft Visual Studio 9.0VCbinvcvars64.bat to C:Program Files (x86)Microsoft Visual Studio 9.0VCbinamd64vcvarsamd64.bat

Step 2 – Install Python

Install Python 3.8.x from: http://www.python.org/download/

Remember the specific major and minor version installed, because it will be needed in the next step. In this case, it is "3.8".

Note: 64-bit: Just download and install the 64-bit version of python.

Step 3 – Install Win32py extensions

Download pywin32 from: http://sourceforge.net/projects/pywin32/files/

- Click on "pywin32" folder
- Click on the first folder (in this case, Build 218, maybe newer when you try)
- Choose the file ending with ".win32-py3.x.exe" -> x being the minor version of Python you installed (in this case, 7) When writing this guide, the file was: http://sourceforge.net/projects/pywin32/files/pywin32/Build%20218/pywin32-218.win-amd64-py3.8.exe/download

  Note: 64-bit: Download and install the 64-bit version. At the time of writing you can find this at: http://sourceforge.net/projects/pywin32/files/pywin32/Build%20218/pywin32-218.win-amd64-py3.8.exe/download

Step 4 – Python BIN

Add Python BIN folder to the path

You have to add the Python folder to the path, you can do it manually (editing "PATH" environment variable) or using Windows Support Tools that came preinstalled in Vista/7 and can be installed in Windows XP.

- Using support tools on WINDOWS XP: If you use Windows XP you can install them using Windows XP CD and navigating to SUPPORTTOOLS. There, execute Setup.EXE (not MSI). Afterwards, open a CMD and type:

  ```
  SETX PATH "%PATH%;[your-python-path]" -M
  ```

  Close CMD (the path variable will be updated then)

- Using support tools on WINDOWS Vista/7:

  Open a CMD and type:

  ```
  SETX PATH "%PATH%;[your-python-path]" /M
  ```

  Please substitute [your-python-path] with your Python installation path. Typically: C:\Python38

Step 5 – Kallithea folder structure

Create a Kallithea folder structure

This is only a example to install Kallithea, you can of course change it. However, this guide will follow the proposed structure, so please later adapt the paths if you change them. My recommendation is to use folders with NO SPACES. But you can try if you are brave...

Create the following folder structure:

```
C:\Kallithea
C:\Kallithea\Bin
C:\Kallithea\Env
C:\Kallithea\Repos
```

Step 6 – Install virtualenv

Create a virtual Python environment in C:\Kallithea\Env (or similar). To do so, open a CMD (Python Path should be included in Step3), and write:

```
python3 -m venv C:\Kallithea\Env
```

Step 7 – Install Kallithea

Finally, install Kallithea

Close previously opened command prompt/s, and open a Visual Studio 2008 Command Prompt (**IMPORTANT!!**). To do so, go to Start Menu, and then open "Microsoft Visual C++ 2008 Express Edition" -> "Visual Studio Tools" -> "Visual Studio 2008 Command Prompt"

Note: 64-bit: For 64-bit you need to modify the shortcut that is used to start the Visual Studio 2008 Command Prompt. Use right-mouse click to open properties.

Change commandline from:

```
%comspec% /k ""C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\vcvarsall.bat""␣
↪x86
```

to:

```
%comspec% /k ""C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\vcvarsall.bat""␣
↪amd64
```

In that CMD (loaded with VS2008 PATHs) type:

```
cd C:\Kallithea\Env\Scripts (or similar)
activate
pip install --upgrade pip setuptools
```

The prompt will change into "(Env) C:\Kallithea\Env\Scripts" or similar (depending of your folder structure). Then type:

```
pip install kallithea
```

(long step, please wait until fully complete)

Some warnings will appear, don't worry as they are normal.

Step 8 – Configuring Kallithea

steps taken from http://packages.python.org/Kallithea/setup.html

You have to use the same Visual Studio 2008 command prompt as Step7, so if you closed it reopen it following the same commands (including the "activate" one). When ready, just type:

```
cd C:\Kallithea\Bin
kallithea-cli config-create my.ini
```

Then, you must edit my.ini to fit your needs (network address and port, mail settings, database, whatever). I recommend using NotePad++ (free) or similar text editor, as it handles well the EndOfLine character differences between Unix and Windows (http://notepad-plus-plus.org/)

For the sake of simplicity lets run it with the default settings. After your edits (if any), in the previous Command Prompt, type:

```
kallithea-cli db-create -c my.ini
```

Warning: This time a *new* database will be installed. You must follow a different process to later *upgrade* to a newer Kallithea version.

The script will ask you for confirmation about creating a NEW database, answer yes (y) The script will ask you for repository path, answer C:\Kallithea\Repos (or similar) The script will ask you for admin username and password, answer "admin" + "123456" (or whatever you want) The script will ask you for admin mail, answer "admin@xxxx.com" (or whatever you want)

If you make some mistake and the script does not end, don't worry, start it again.

Step 9 – Running Kallithea

In the previous command prompt, being in the C:\Kallithea\Bin folder, just type:

```
gearbox serve -c my.ini
```

Open yout web server, and go to http://127.0.0.1:5000

It works!! :-)

Remark: If it does not work first time, just Ctrl-C the CMD process and start it again. Don't forget the "http://" in Internet Explorer

What this Guide does not cover:

- Installing Celery
- Running Kallithea as Windows Service. You can investigate here:
  - http://pypi.python.org/pypi/wsgisvc
  - http://ryrobes.com/python/running-python-scripts-as-a-windows-service/
  - http://wiki.pylonshq.com/display/pylonscookbook/How+to+run+Pylons+as+a+Windows+service
- Using Apache. You can investigate here:
  - https://groups.google.com/group/rhodecode/msg/c433074e813ffdc4

Warning: This section is outdated and needs updating for Python 3.

2.5 Installing Kallithea on Microsoft Internet Information Services (IIS)

The following is documented using IIS 7/8 terminology. There should be nothing preventing you from applying this on IIS 6 well.

Note: Installing Kallithea under IIS can enable Single Sign-On to the Kallithea web interface from web browsers that can authenticate to the web server. (As an alternative to IIS, SSO is also possible with for example Apache and mod_sspi.)

Mercurial and Git do however by default not support SSO on the client side and will still require some other kind of authentication. (An extension like hgssoauthentication might solve that.)

Note: For the best security, it is strongly recommended to only host the site over a secure connection, e.g. using TLS.

2.5.1 Prerequisites

Apart from the normal requirements for Kallithea, it is also necessary to get an ISAPI-WSGI bridge module, e.g. isapi-wsgi.

2.5.2 Installation

The following assumes that your Kallithea is at `c:\inetpub\kallithea`, and will be served from the root of its own website. The changes to serve it in its own virtual folder will be noted where appropriate.

Application pool

Make sure that there is a unique application pool for the Kallithea application with an identity that has read access to the Kallithea distribution.

The application pool does not need to be able to run any managed code. If you are using a 32-bit Python installation, then you must enable 32-bit program in the advanced settings for the application pool; otherwise Python will not be able to run on the website and neither will Kallithea.

Note: The application pool can be the same as an existing application pool, as long as the Kallithea requirements are met by the existing pool.

ISAPI handler

The ISAPI handler can be generated using:

```
kallithea-cli iis-install -c my.ini --virtualdir=/
```

This will generate a `dispatch.py` file in the current directory that contains the necessary components to finalize an installation into IIS. Once this file has been generated, it is necessary to run the following command due to the way that ISAPI-WSGI is made:

```
python3 dispatch.py install
```

This accomplishes two things: generating an ISAPI compliant DLL file, `_dispatch.dll`, and installing a script map handler into IIS for the `--virtualdir` specified above pointing to `_dispatch.dll`.

The ISAPI handler is registered to all file extensions, so it will automatically be the one handling all requests to the specified virtual directory. When the website starts the ISAPI handler, it will start a thread pool managed wrapper around the middleware WSGI handler that Kallithea runs within and each HTTP request to the site will be processed through this logic henceforth.

Authentication with Kallithea using IIS authentication modules

The recommended way to handle authentication with Kallithea using IIS is to let IIS handle all the authentication and just pass it to Kallithea.

Note: As an alternative without SSO, you can also use LDAP authentication with Active Directory, see *LDAP Authentication*.

To move responsibility into IIS from Kallithea, we need to configure Kallithea to let external systems handle authentication and then let Kallithea create the user automatically. To do this, access the administration's authentication page and enable the `kallithea.lib.auth_modules.auth_container` plugin. Once it is added, enable it with the `REMOTE_USER` header and check *Clean username*. Finally, save the changes on this page.

Switch to the administration's permissions page and disable anonymous access, otherwise Kallithea will not attempt to use the authenticated user name. By default, Kallithea will populate the list of users lazily as they log in. Either disable external auth account activation and ensure that you pre-populate the user database with an external tool, or set it to *Automatic activation of external account*. Finally, save the changes.

The last necessary step is to enable the relevant authentication in IIS, e.g. Windows authentication.

2.5.3 Troubleshooting

Typically, any issues in this setup will either be entirely in IIS or entirely in Kallithea (or Kallithea's WSGI middleware). Consequently, two different options for finding issues exist: IIS' failed request tracking which is great at finding issues until they exist inside Kallithea, at which point the ISAPI-WSGI wrapper above uses `win32traceutil`, which is part of `pywin32`.

In order to dump output from WSGI using `win32traceutil` it is sufficient to type the following in a console window:

```
python3 -m win32traceutil
```

and any exceptions occurring in the WSGI layer and below (i.e. in the Kallithea application itself) that are uncaught, will be printed here complete with stack traces, making it a lot easier to identify issues.

2.6 Installation and setup using Puppet

The whole installation and setup process of Kallithea can be simplified by using Puppet and the rauch/kallithea Puppet module. This is especially useful for getting started quickly, without having to deal with all the Python specialities.

Note: The following instructions assume you are not familiar with Puppet at all. If this is not the case, you should probably skip directly to the Kallithea Puppet module documentation.

2.6.1 Installing Puppet

This installation variant requires a Unix/Linux type server with Puppet 3.0+ installed. Many major distributions have Puppet in their standard repositories. Thus, you will probably be ready to go by running, e.g. `apt-get install puppet` or `yum install puppet`, depending on your distro's favoured package manager. Afterwards, check the Puppet version by running `puppet --version` and ensure you have at least 3.0.

If your distribution does not provide Puppet packages or you need a newer version, please see the Puppet Reference Manual for instructions on how to install Puppet on your target platform.

2.6.2 Installing the Puppet module

To install the latest version of the Kallithea Puppet module from the Puppet Forge, run the following as `root`:

```
puppet module install rauch/kallithea
```

This will install both the Kallithea Puppet module and its dependency modules.

Warning: Be aware that Puppet can do all kinds of things to your systems. Third-party modules (like the `kallithea` module) may run arbitrary commands on your system (most of the time as the `root` user), so do not apply them on production machines if you don't know what you are doing. Instead, use a test system (e.g. a virtual machine) for evaluation purposes.

2.6.3 Applying the module

To trigger the actual installation process, we have to *apply* the `kallithea` Puppet class, which is provided by the module we have just installed, to our system. For this, create a file named e.g. `kallithea.pp`, a *Puppet manifest*, with the following content:

```
class { 'kallithea':
  seed_db    => true,
  manage_git => true,
}
```

To apply the manifest, simply run the following (preferably as root):

```
puppet apply kallithea.pp
```

This will basically run through the usual Kallithea *Installation on Unix/Linux* and *Setup* steps, as documented. Consult the module documentation for details on what the module affects. You can also do a *dry run* by adding the `--noop` option to the command.

2.6.4 Using parameters for customizing the setup process

The `kallithea` Puppet class provides a number of parameters for customizing the setup process. You have seen the usage of the `seed_db` parameter in the *example above*, but there are more. For example, you can specify the

installation directory, the name of the user under which Kallithea gets installed, the initial admin password, etc. Notably, you can provide arbitrary modifications to Kallithea's configuration file by means of the `config_hash` parameter.

Parameters, which have not been set explicitly, will be set to default values, which are defined inside the `kallithea` Puppet module. For example, if you just stick to the defaults as in the *example above*, you will end up with a Kallithea instance, which

- is installed in `/srv/kallithea`, owned by the user `kallithea`
- uses the Kallithea default configuration
- uses the admin user `admin` with password `adminpw`
- is started automatically and enabled on boot

As of Kallithea 0.3.0, this in particular means that Kallithea will use an SQLite database and listen on `http://localhost:5000`.

See also the full parameters list for more information.

2.6.5 Making your Kallithea instance publicly available

If you followed the instructions above, the Kallithea instance will be listening on `http://localhost:5000` and therefore not publicly available. There are several ways to change this.

The direct way

The simplest setup is to instruct Kallithea to listen on another IP address and/or port by using the `config_hash` parameter of the Kallithea Puppet class. For example, assume we want to listen on all interfaces on port 80:

```
class { 'kallithea':
  seed_db => true,
  config_hash => {
    "server:main" => {
      'host' => '0.0.0.0',
      'port' => '80',
    }
  }
}
```

Using Apache as reverse proxy

In a more advanced setup, you might instead want use a full-blown web server like Apache HTTP Server as the public web server, configured such that requests are internally forwarded to the local Kallithea instance (a so called *reverse proxy setup*). This can be easily done with Puppet as well:

First, install the puppetlabs/apache Puppet module as above by running the following as root:

```
puppet module install puppetlabs/apache
```

Then, append the following to your manifest:

```
include apache

apache::vhost { 'kallithea.example.com':
```

(continues on next page)

(continued from previous page)

```
  docroot             => '/var/www/html',
  manage_docroot      => false,
  port                => 80,
  proxy_preserve_host => true,
  proxy_pass          => [
    {
      path => '/',
      url  => 'http://localhost:5000/',
    },
  ],
}
```

Applying the resulting manifest will install the Apache web server and setup a virtual host acting as a reverse proxy for your local Kallithea instance.

2.7 Upgrading Kallithea

This describes the process for upgrading Kallithea, independently of the Kallithea installation method.

Note: If you are upgrading from a RhodeCode installation, you must first install Kallithea 0.3.2 and follow the instructions in the 0.3.2 README to perform a one-time conversion of the database from RhodeCode to Kallithea, before upgrading to the latest version of Kallithea.

2.7.1 1. Stop the Kallithea web application

This step depends entirely on the web server software used to serve Kallithea, but in any case, Kallithea should not be running during the upgrade.

Note: If you're using Celery, make sure you stop all instances during the upgrade.

2.7.2 2. Create a backup of both database and configuration

You are of course strongly recommended to make backups regularly, but it is *especially* important to make a full database and configuration backup before performing a Kallithea upgrade.

Back up your configuration

Make a copy of your Kallithea configuration (`.ini`) file.

If you are using custom *extensions*, you should also make a copy of the `extensions.py` file.

Back up your database

If using SQLite, simply make a copy of the Kallithea database (`.db`) file.

If using PostgreSQL, please consult the documentation for the `pg_dump` utility.

If using MariaDB/MySQL, please consult the documentation for the `mysqldump` utility.

Look for `sqlalchemy.url` in your configuration file to determine database type, settings, location, etc. If you were running Kallithea 0.3.x or older, this was `sqlalchemy.db1.url`.

2.7.3 3. Activate or recreate the Kallithea virtual environment (if any)

Note: If you did not install Kallithea in a virtual environment, skip this step.

For major upgrades, e.g. from 0.3.x to 0.4.x, it is recommended to create a new virtual environment, rather than reusing the old. For minor upgrades, e.g. within the 0.4.x range, this is not really necessary (but equally fine).

To create a new virtual environment, please refer to the appropriate installation page for details. After creating and activating the new virtual environment, proceed with the rest of the upgrade process starting from the next section.

To reuse the same virtual environment, first activate it, then verify that you are using the correct environment by running:

```
pip freeze
```

This will list all packages installed in the current environment. If Kallithea isn't listed, deactivate the environment and then activate the correct one, or recreate a new environment. See the appropriate installation page for details.

2.7.4 4. Install new version of Kallithea

Please refer to the instructions for the installation method you originally used to install Kallithea.

If you originally installed using pip, it is as simple as:

```
pip install --upgrade kallithea
```

If you originally installed from version control, assuming you did not make private changes (in which case you should adapt the instructions accordingly):

```
cd my-kallithea-clone
hg parent   # make a note of the original revision
hg pull
hg update
hg parent   # make a note of the new revision
pip install --upgrade -e .
```

2.7.5 5. Upgrade your configuration

Run the following command to create a new configuration (`.ini`) file:

```
kallithea-cli config-create new.ini
```

Then compare it with your old config file and copy over the required configuration values from the old to the new file.

Note: Please always make sure your `.ini` files are up to date. Errors can often be caused by missing parameters added in new versions.

2.7.6 6. Upgrade your database

Note: If you are *downgrading* Kallithea, you should perform the database migration step *before* installing the older version. (That is, always perform migrations using the most recent of the two versions you're migrating between.)

First, run the following command to see your current database version:

```
alembic -c new.ini current
```

Typical output will be something like "9358dc3d6828 (head)", which is the current Alembic database "revision ID". Write down the entire output for troubleshooting purposes.

The output will be empty if you're upgrading from Kallithea 0.3.x or older. That's expected. If you get an error that the config file was not found or has no `[alembic]` section, see the next section.

Next, if you are performing an *upgrade*: Run the following command to upgrade your database to the current Kallithea version:

```
alembic -c new.ini upgrade head
```

If you are performing a *downgrade*: Run the following command to downgrade your database to the given version:

```
alembic -c new.ini downgrade 0.4
```

Alembic will show the necessary migrations (if any) as it executes them. If no "ERROR" is displayed, the command was successful.

Should an error occur, the database may be "stranded" half-way through the migration, and you should restore it from backup.

Enabling old Kallithea config files for Alembic use

Kallithea configuration files created before the introduction of Alembic (i.e. predating Kallithea 0.4) need to be updated for use with Alembic. Without this, Alembic will fail with an error like this:

```
FAILED: No config file 'my.ini' found, or file has no '[alembic]' section
```

Note: If you followed this upgrade guide correctly, you will have created a new configuration file in section *Upgrading your configuration*. When calling Alembic, make sure to use this new config file. In this case, you should not get any errors and the below manual steps should not be needed.

If Alembic complains specifically about a missing `alembic.ini`, it is likely because you did not specify a config file using the `-c` option. On the other hand, if the mentioned config file actually exists, you need to append the following lines to it:

```
[alembic]
script_location = kallithea:alembic
```

Your config file should now work with Alembic.

2.7.7 7. Prepare the front-end

Starting with Kallithea 0.4, external front-end dependencies are no longer shipped but need to be downloaded and/or generated at installation time. Run the following command:

```
kallithea-cli front-end-build
```

2.7.8 8. Rebuild the Whoosh full-text index

It is recommended that you rebuild the Whoosh index after upgrading since new Whoosh versions can introduce incompatible index changes.

2.7.9 9. Start the Kallithea web application

This step once again depends entirely on the web server software used to serve Kallithea.

If you were running Kallithea 0.3.x or older and were using `paster serve my.ini` before, then the corresponding command in Kallithea 0.4 and later is:

```
gearbox serve -c new.ini
```

Before starting the new version of Kallithea, you may find it helpful to clear out your log file so that new errors are readily apparent.

Note: If you're using Celery, make sure you restart all instances of it after upgrade.

2.7.10 10. Reinstall internal Git repository hooks

It is possible that an upgrade involves changes to the Git hooks installed by Kallithea. As these hooks are created inside the repositories on the server filesystem, they are not updated automatically when upgrading Kallithea itself.

To update the hooks of your Git repositories, run:

```
kallithea-cli repo-scan -c my.ini --install-git-hooks
```

Watch out for warnings like `skipping overwriting hook file X`, then fix it and rerun, or consider using `--overwrite-git-hooks` instead.

Or:

- Go to *Admin* > *Settings* > *Remap and Rescan*
- Select the checkbox *Install Git hooks*
- Click the button *Rescan repositories*

Note: Kallithea does not use hooks on Mercurial repositories. This step is thus not necessary if you only have Mercurial repositories.

Setup and configuration

2.8 Setup

2.8.1 Setting up a Kallithea instance

Some further details to the steps mentioned in the overview.

Create low level configuration file

First, you will need to create a Kallithea configuration file. The configuration file is a `.ini` file that contains various low level settings for Kallithea, e.g. configuration of how to use database, web server, email, and logging.

Change to the desired directory (such as `/srv/kallithea`) as the right user and run the following command to create the file `my.ini` in the current directory:

```
kallithea-cli config-create my.ini http_server=waitress
```

To get a good starting point for your configuration, specify the http server you intend to use. It can be `waitress`, `gearbox`, `gevent`, `gunicorn`, or `uwsgi`. (Apache `mod_wsgi` will not use this configuration file, and it is fine to keep the default http_server configuration unused. `mod_wsgi` is configured using `httpd.conf` directives and a WSGI wrapper script.)

Extra custom settings can be specified like:

```
kallithea-cli config-create my.ini host=8.8.8.8 "[handler_console]" formatter=color_
↪formatter
```

Populate the database

Next, you need to create the databases used by Kallithea. Kallithea currently supports PostgreSQL, SQLite and MariaDB/MySQL databases. It is recommended to start out using SQLite (the default) and move to PostgreSQL if it becomes a bottleneck or to get a "proper" database. MariaDB/MySQL is also supported.

For PostgreSQL, run `pip install psycopg2` to get the database driver. Make sure the PostgreSQL server is initialized and running. Make sure you have a database user with password authentication with permissions to create databases - for example by running:

```
sudo -u postgres createuser 'kallithea' --pwprompt --createdb
```

For MariaDB/MySQL, run `pip install mysqlclient` to get the `MySQLdb` database driver. Make sure the database server is initialized and running. Make sure you have a database user with password authentication with permissions to create the database - for example by running:

```
echo 'CREATE USER "kallithea"@"localhost" IDENTIFIED BY "password"' | sudo -u mysql␣
↪mysql
echo 'GRANT ALL PRIVILEGES ON `kallithea`.* TO "kallithea"@"localhost"' | sudo -u␣
↪mysql mysql
```

Check and adjust `sqlalchemy.url` in your `my.ini` configuration file to use this database.

Create the database, tables, and initial content by running the following command:

```
kallithea-cli db-create -c my.ini
```

This will first prompt you for a "root" path. This "root" path is the location where Kallithea will store all of its repositories on the current machine. This location must be writable for the running Kallithea application. Next, `db-create` will prompt you for a username and password for the initial admin account it sets up for you.

The `db-create` values can also be given on the command line. Example:

```
kallithea-cli db-create -c my.ini --user=nn --password=secret --email=nn@example.com -
↪-repos=/srv/repos
```

The `db-create` command will create all needed tables and an admin account. When choosing a root path you can either use a new empty location, or a location which already contains existing repositories. If you choose a location which contains existing repositories Kallithea will add all of the repositories at the chosen location to its database. (Note: make sure you specify the correct path to the root).

Note: It is also possible to use an existing database. For example, when using PostgreSQL without granting general createdb privileges to the PostgreSQL kallithea user, set `sqlalchemy.url = postgresql://kallithea:password@localhost/kallithea` and create the database like:

```
sudo -u postgres createdb 'kallithea' --owner 'kallithea'
kallithea-cli db-create -c my.ini --reuse
```

Running

You are now ready to use Kallithea. To run it using a gearbox web server, simply execute:

```
gearbox serve -c my.ini
```

- This command runs the Kallithea server. The web app should be available at http://127.0.0.1:5000. The IP address and port is configurable via the configuration file created in the previous step.
- Log in to Kallithea using the admin account created when running `db-create`.
- The default permissions on each repository is read, and the owner is admin. Remember to update these if needed.
- In the admin panel you can toggle LDAP, anonymous, and permissions settings, as well as edit more advanced options on users and repositories.

2.8.2 Internationalization (i18n support)

The Kallithea web interface is automatically displayed in the user's preferred language, as indicated by the browser. Thus, different users may see the application in different languages. If the requested language is not available (because the translation file for that language does not yet exist or is incomplete), English is used.

If you want to disable automatic language detection and instead configure a fixed language regardless of user preference, set `i18n.enabled = false` and specify another language by setting `i18n.lang` in the Kallithea configuration file.

2.8.3 Using Kallithea with SSH

Kallithea supports repository access via SSH key based authentication. This means:

- repository URLs like `ssh://kallithea@example.com/name/of/repository`

- all network traffic for both read and write happens over the SSH protocol on port 22, without using HTTP/HTTPS nor the Kallithea WSGI application
- encryption and authentication protocols are managed by the system's `sshd` process, with all users using the same Kallithea system user (e.g. `kallithea`) when connecting to the SSH server, but with users' public keys in the Kallithea system user's *.ssh/authorized_keys* file granting each user sandboxed access to the repositories.
- users and admins can manage SSH public keys in the web UI
- in their SSH client configuration, users can configure how the client should control access to their SSH key - without passphrase, with passphrase, and optionally with passphrase caching in the local shell session (`ssh-agent`). This is standard SSH functionality, not something Kallithea provides or interferes with.
- network communication between client and server happens in a bidirectional stateful stream, and will in some cases be faster than HTTP/HTTPS with several stateless round-trips.

Note: At this moment, repository access via SSH has been tested on Unix only. Windows users that care about SSH are invited to test it and report problems, ideally contributing patches that solve these problems.

Users and admins can upload SSH public keys (e.g. `.ssh/id_rsa.pub`) through the web interface. The server's `.ssh/authorized_keys` file is automatically maintained with an entry for each SSH key. Each entry will tell `sshd` to run `kallithea-cli` with the `ssh-serve` sub-command and the right Kallithea user ID when encountering the corresponding SSH key.

To enable SSH repository access, Kallithea must be configured with the path to the `.ssh/authorized_keys` file for the Kallithea user, and the path to the `kallithea-cli` command. Put something like this in the `.ini` file:

```
ssh_enabled = true
ssh_authorized_keys = /home/kallithea/.ssh/authorized_keys
kallithea_cli_path = /srv/kallithea/venv/bin/kallithea-cli
```

The SSH service must be running, and the Kallithea user account must be active (not necessarily with password access, but public key access must be enabled), all file permissions must be set as sshd wants it, and `authorized_keys` must be writeable by the Kallithea user.

Note: The `authorized_keys` file will be rewritten from scratch on each update. If it already exists with other data, Kallithea will not overwrite the existing `authorized_keys`, and the server process will instead throw an exception. The system administrator thus cannot ssh directly to the Kallithea user but must use su/sudo from another account.

If `/home/kallithea/.ssh/` (the directory of the path specified in the `ssh_authorized_keys` setting of the `.ini` file) does not exist as a directory, Kallithea will attempt to create it. If that path exists but is *not* a directory, or is not readable-writable-executable by the server process, the server process will raise an exception each time it attempts to write the `authorized_keys` file.

Note: It is possible to configure the SSH server to look for authorized keys in multiple files, for example reserving `ssh/authorized_keys` to be used for normal SSH and with Kallithea using `.ssh/authorized_keys_kallithea`. In `/etc/ssh/sshd_config` set `AuthorizedKeysFile .ssh/authorized_keys .ssh/authorized_keys_kallithea` and restart sshd, and in `my.ini` set `ssh_authorized_keys = /home/kallithea/.ssh/authorized_keys_kallithea`. Note that this new location will apply to all system users, and that multiple entries for the same SSH key will shadow each other.

Warning: The handling of SSH access is steered directly by the command specified in the `authorized_keys` file. There is no interaction with the web UI. Once SSH access is correctly configured and enabled, it will work regardless of whether the Kallithea web process is actually running. Hence, if you want to perform repository or server maintenance and want to fully disable all access to the repositories, disable SSH access by setting `ssh_enabled = false` in the correct `.ini` file (i.e. the `.ini` file specified in the `authorized_keys` file.)

The `authorized_keys` file can be updated manually with `kallithea-cli ssh-update-authorized-keys -c my.ini`. This command is not needed in normal operation but is for example useful after changing SSH-related settings in the `.ini` file or renaming that file. (The path to the `.ini` file is used in the generated `authorized_keys` file).

2.8.4 Setting up Whoosh full text search

Kallithea provides full text search of repositories using Whoosh.

For an incremental index build, run:

```
kallithea-cli index-create -c my.ini
```

For a full index rebuild, run:

```
kallithea-cli index-create -c my.ini --full
```

The `--repo-location` option allows the location of the repositories to be overridden; usually, the location is retrieved from the Kallithea database.

The `--index-only` option can be used to limit the indexed repositories to a comma-separated list:

```
kallithea-cli index-create -c my.ini --index-only=vcs,kallithea
```

To keep your index up-to-date it is necessary to do periodic index builds; for this, it is recommended to use a crontab entry. Example:

```
0  3  *  *  *  /path/to/virtualenv/bin/kallithea-cli index-create -c /path/to/
↪kallithea/my.ini
```

When using incremental mode (the default), Whoosh will check the last modification date of each file and add it to be reindexed if a newer file is available. The indexing daemon checks for any removed files and removes them from index.

If you want to rebuild the index from scratch, you can use the `-f` flag as above, or in the admin panel you can check the "build from scratch" checkbox.

2.8.5 Integration with issue trackers

Kallithea provides a simple integration with issue trackers. It's possible to define a regular expression that will match an issue ID in commit messages, and have that replaced with a URL to the issue.

This is achieved with following three variables in the ini file:

```
issue_pat = #(\d+)
issue_server_link = https://issues.example.com/{repo}/issue/\1
issue_sub =
```

`issue_pat` is the regular expression describing which strings in commit messages will be treated as issue references. The expression can/should have one or more parenthesized groups that can later be referred to in `issue_server_link` and `issue_sub` (see below). If you prefer, named groups can be used instead of simple parenthesized groups.

If the pattern should only match if it is preceded by whitespace, add the following string before the actual pattern: `(?:^|(?<=\s))`. If the pattern should only match if it is followed by whitespace, add the following string after the actual pattern: `(?:$|(?=\s))`. These expressions use lookbehind and lookahead assertions of the Python regular expression module to avoid the whitespace to be part of the actual pattern, otherwise the link text will also contain that whitespace.

Matched issue references are replaced with the link specified in `issue_server_link`, in which any backreferences are resolved. Backreferences can be `\1`, `\2`, ... or for named groups `\g<groupname>`. The special token `{repo}` is replaced with the full repository path (including repository groups), while token `{repo_name}` is replaced with the repository name (without repository groups).

The link text is determined by `issue_sub`, which can be a string containing backreferences to the groups specified in `issue_pat`. If `issue_sub` is empty, then the text matched by `issue_pat` is used verbatim.

The example settings shown above match issues in the format `#<number>`. This will cause the text `#300` to be transformed into a link:

```
<a href="https://issues.example.com/example_repo/issue/300">#300</a>
```

The following example transforms a text starting with either of 'pullrequest', 'pull request' or 'PR', followed by an optional space, then a pound character (#) and one or more digits, into a link with the text 'PR #' followed by the digits:

```
issue_pat = (pullrequest|pull request|PR) ?#(\d+)
issue_server_link = https://issues.example.com/\2
issue_sub = PR #\2
```

The following example demonstrates how to require whitespace before the issue reference in order for it to be recognized, such that the text `issue#123` will not cause a match, but `issue #123` will:

```
issue_pat = (?:^|(?<=\s))#(\d+)
issue_server_link = https://issues.example.com/\1
issue_sub =
```

If needed, more than one pattern can be specified by appending a unique suffix to the variables. For example, also demonstrating the use of named groups:

```
issue_pat_wiki = wiki-(?P<pagename>\S+)
issue_server_link_wiki = https://wiki.example.com/\g<pagename>
issue_sub_wiki = WIKI-\g<pagename>
```

With these settings, wiki pages can be referenced as wiki-some-id, and every such reference will be transformed into:

```
<a href="https://wiki.example.com/some-id">WIKI-some-id</a>
```

Refer to the Python regular expression documentation for more details about the supported syntax in `issue_pat`, `issue_server_link` and `issue_sub`.

2.8.6 Hook management

Custom Mercurial hooks can be managed in a similar way to that used in `.hgrc` files. To manage hooks, choose *Admin* > *Settings* > *Hooks*.

To add another custom hook simply fill in the first textbox with `<name>.<hook_type>` and the second with the hook path. Example hooks can be found in `kallithea.lib.hooks`.

Kallithea will also use some hooks internally. They cannot be modified, but some of them can be enabled or disabled in the *VCS* section.

Kallithea does not actively support custom Git hooks, but hooks can be installed manually in the file system. Kallithea will install and use the `post-receive` Git hook internally, but it will then invoke `post-receive-custom` if present.

2.8.7 Changing default encoding

By default, Kallithea uses UTF-8 encoding. This is configurable as `default_encoding` in the .ini file. This affects many parts in Kallithea including user names, filenames, and encoding of commit messages. In addition Kallithea can detect if the `chardet` library is installed. If `chardet` is detected Kallithea will fallback to it when there are encode/decode errors.

The Mercurial encoding is configurable as `hgencoding`. It is similar to setting the `HGENCODING` environment variable, but will override it.

2.8.8 Celery configuration

Kallithea can use the distributed task queue system Celery to run tasks like cloning repositories or sending emails.

Kallithea will in most setups work perfectly fine out of the box (without Celery), executing all tasks in the web server process. Some tasks can however take some time to run and it can be better to run such tasks asynchronously in a separate process so the web server can focus on serving web requests.

For installation and configuration of Celery, see the Celery documentation. Note that Celery requires a message broker service like RabbitMQ (recommended) or Redis.

The use of Celery is configured in the Kallithea ini configuration file. To enable it, simply set:

```
use_celery = true
```

and add or change the `celery.*` configuration variables.

Configuration settings are prefixed with 'celery.', so for example setting *broker_url* in Celery means setting *celery.broker_url* in the configuration file.

To start the Celery process, run:

```
kallithea-cli celery-run -c my.ini
```

Extra options to the Celery worker can be passed after `--` - see `-- -h` for more info.

Note: Make sure you run this command from the same virtualenv, and with the same user that Kallithea runs.

2.8.9 Proxy setups

When Kallithea is processing HTTP requests from a user, it will see and use some of the basic properties of the connection, both at the TCP/IP level and at the HTTP level. The WSGI server will provide this information to Kallithea in the "environment".

In some setups, a proxy server will take requests from users and forward them to the actual Kallithea server. The proxy server will thus be the immediate client of the Kallithea WSGI server, and Kallithea will basically see it as such. To make sure Kallithea sees the request as it arrived from the client to the proxy server, the proxy server must be configured to somehow pass the original information on to Kallithea, and Kallithea must be configured to pick that information up and trust it.

Kallithea will by default rely on its WSGI server to provide the IP of the client in the WSGI environment as `REMOTE_ADDR`, but it can be configured to get it from an HTTP header that has been set by the proxy server. For example, if the proxy server puts the client IP in the `X-Forwarded-For` HTTP header, set:

```
remote_addr_variable = HTTP_X_FORWARDED_FOR
```

Kallithea will by default rely on finding the protocol (`http` or `https`) in the WSGI environment as `wsgi.url_scheme`. If the proxy server puts the protocol of the client request in the `X-Forwarded-Proto` HTTP header, Kallithea can be configured to trust that header by setting:

```
url_scheme_variable = HTTP_X_FORWARDED_PROTO
```

2.8.10 HTTPS support

Kallithea will by default generate URLs based on the WSGI environment.

Alternatively, you can use some special configuration settings to control directly which scheme/protocol Kallithea will use when generating URLs:

- With `url_scheme_variable` set, the scheme will be taken from that HTTP header.
- With `force_https = true`, the scheme will be seen as `https`.
- With `use_htsts = true`, Kallithea will set `Strict-Transport-Security` when using https.

2.8.11 Nginx virtual host example

Sample config for Nginx using proxy:

```
upstream kallithea {
    server 127.0.0.1:5000;
    # add more instances for load balancing
    #server 127.0.0.1:5001;
    #server 127.0.0.1:5002;
}

## gist alias
server {
   listen          443;
   server_name     gist.example.com;
   access_log      /var/log/nginx/gist.access.log;
   error_log       /var/log/nginx/gist.error.log;

   ssl on;
   ssl_certificate     gist.your.kallithea.server.crt;
   ssl_certificate_key gist.your.kallithea.server.key;

   ssl_session_timeout 5m;

   ssl_protocols SSLv3 TLSv1;
```

(continues on next page)

(continued from previous page)

```
   ssl_ciphers DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA:EDH-RSA-DES-CBC3-SHA:AES256-
↪SHA:DES-CBC3-SHA:AES128-SHA:RC4-SHA:RC4-MD5;
   ssl_prefer_server_ciphers on;

   rewrite ^/(.+)$ https://kallithea.example.com/_admin/gists/$1;
   rewrite (.*)    https://kallithea.example.com/_admin/gists;
}

server {
   listen          443;
   server_name     kallithea.example.com
   access_log      /var/log/nginx/kallithea.access.log;
   error_log       /var/log/nginx/kallithea.error.log;

   ssl on;
   ssl_certificate     your.kallithea.server.crt;
   ssl_certificate_key your.kallithea.server.key;

   ssl_session_timeout 5m;

   ssl_protocols SSLv3 TLSv1;
   ssl_ciphers DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA:EDH-RSA-DES-CBC3-SHA:AES256-
↪SHA:DES-CBC3-SHA:AES128-SHA:RC4-SHA:RC4-MD5;
   ssl_prefer_server_ciphers on;

   ## uncomment root directive if you want to serve static files by nginx
   ## requires static_files = false in .ini file
   #root /srv/kallithea/kallithea/kallithea/public;
   include         /etc/nginx/proxy.conf;
   location / {
        try_files $uri @kallithea;
   }

   location @kallithea {
        proxy_pass      http://127.0.0.1:5000;
   }

}
```

Here's the proxy.conf. It's tuned so it will not timeout on long pushes or large pushes:

```
proxy_redirect              off;
proxy_set_header            Host $host;
## needed for container auth
#proxy_set_header            REMOTE_USER $remote_user;
#proxy_set_header            X-Forwarded-User $remote_user;
proxy_set_header            X-Url-Scheme $scheme;
proxy_set_header            X-Host $http_host;
proxy_set_header            X-Real-IP $remote_addr;
proxy_set_header            X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header            Proxy-host $proxy_host;
proxy_buffering             off;
proxy_connect_timeout       7200;
proxy_send_timeout          7200;
proxy_read_timeout          7200;
proxy_buffers               8 32k;
client_max_body_size        1024m;
```

(continues on next page)

(continued from previous page)

```
client_body_buffer_size     128k;
large_client_header_buffers 8 64k;
```

2.8.12 Apache virtual host reverse proxy example

Here is a sample configuration file for Apache using proxy:

```
<VirtualHost *:80>
        ServerName kallithea.example.com

        <Proxy *>
          # For Apache 2.4 and later:
          Require all granted

          # For Apache 2.2 and earlier, instead use:
          # Order allow,deny
          # Allow from all
        </Proxy>

        #important !
        #Directive to properly generate url (clone url) for Kallithea
        ProxyPreserveHost On

        #kallithea instance
        ProxyPass / http://127.0.0.1:5000/
        ProxyPassReverse / http://127.0.0.1:5000/

        #to enable https use line below
        #SetEnvIf X-Url-Scheme https HTTPS=1
</VirtualHost>
```

Additional tutorial http://pylonsbook.com/en/1.1/deployment.html#using-apache-to-proxy-requests-to-pylons

2.8.13 Apache as subdirectory

Apache subdirectory part:

```
<Location /PREFIX >
  ProxyPass http://127.0.0.1:5000/PREFIX
  ProxyPassReverse http://127.0.0.1:5000/PREFIX
  SetEnvIf X-Url-Scheme https HTTPS=1
</Location>
```

Besides the regular apache setup you will need to add the following line into `[app:main]` section of your .ini file:

```
filter-with = proxy-prefix
```

Add the following at the end of the .ini file:

```
[filter:proxy-prefix]
use = egg:PasteDeploy#prefix
prefix = /PREFIX
```

then change `PREFIX` into your chosen prefix

2.8.14 Apache with mod_wsgi

Alternatively, Kallithea can be set up with Apache under mod_wsgi. For that, you'll need to:

- Install mod_wsgi. If using a Debian-based distro, you can install the package libapache2-mod-wsgi:

```
aptitude install libapache2-mod-wsgi
```

- Enable mod_wsgi:

```
a2enmod wsgi
```

- Add global Apache configuration to tell mod_wsgi that Python only will be used in the WSGI processes and shouldn't be initialized in the Apache processes:

```
WSGIRestrictEmbedded On
```

- Create a WSGI dispatch script, like the one below. The `WSGIDaemonProcess python-home` directive will make sure it uses the right Python Virtual Environment and that paste thus can pick up the right Kallithea application.

```
ini = '/srv/kallithea/my.ini'
from logging.config import fileConfig
fileConfig(ini, {'__file__': ini, 'here': '/srv/kallithea'})
from paste.deploy import loadapp
application = loadapp('config:' + ini)
```

- Add the necessary `WSGI*` directives to the Apache Virtual Host configuration file, like in the example below. Notice that the WSGI dispatch script created above is referred to with the `WSGIScriptAlias` directive. The default locale settings Apache provides for web services are often not adequate, with *C* as the default language and *ASCII* as the encoding. Instead, use the `lang` parameter of `WSGIDaemonProcess` to specify a suitable locale. See also the *Installation overview* section and the WSGIDaemonProcess documentation.

  Apache will by default run as a special Apache user, on Linux systems usually `www-data` or `apache`. If you need to have the repositories directory owned by a different user, use the user and group options to WSGIDaemonProcess to set the name of the user and group.

  Once again, check that all paths are correctly specified.

```
WSGIDaemonProcess kallithea processes=5 threads=1 maximum-requests=100 \
    python-home=/srv/kallithea/venv lang=C.UTF-8
WSGIProcessGroup kallithea
WSGIScriptAlias / /srv/kallithea/dispatch.wsgi
WSGIPassAuthorization On
```

2.8.15 Other configuration files

A number of example init.d scripts can be found in the `init.d` directory of the Kallithea source.

2.9 Authentication setup

Users can be authenticated in different ways. By default, Kallithea uses its internal user database. Alternative authentication methods include LDAP, PAM, Crowd, and container-based authentication.

2.9.1 LDAP Authentication

Kallithea supports LDAP authentication. In order to use LDAP, you have to install the python-ldap package. This package is available via PyPI, so you can install it by running:

```
pip install python-ldap
```

Note: `python-ldap` requires some libraries to be installed on your system, so before installing it check that you have at least the `openldap` and `sasl` libraries.

Choose *Admin* > *Authentication*, click the `kallithea.lib.auth_modules.auth_ldap` button and then *Save*, to enable the LDAP plugin and configure its settings.

Here's a typical LDAP setup:

```
Connection settings
Enable LDAP          = checked
Host                 = host.example.com
Account              = <account>
Password             = <password>
Connection Security  = LDAPS
Certificate Checks   = DEMAND

Search settings
Base DN              = CN=users,DC=host,DC=example,DC=org
LDAP Filter          = (&(objectClass=user)(!(objectClass=computer)))
LDAP Search Scope    = SUBTREE

Attribute mappings
Login Attribute      = uid
First Name Attribute = firstName
Last Name Attribute  = lastName
Email Attribute      = mail
```

If your user groups are placed in an Organisation Unit (OU) structure, the Search Settings configuration differs:

```
Search settings
Base DN              = DC=host,DC=example,DC=org
LDAP Filter          = (&(memberOf=CN=your user group,OU=subunit,OU=unit,DC=host,
↪DC=example,DC=org)(objectClass=user))
LDAP Search Scope    = SUBTREE
```

Enable LDAP [required] Whether to use LDAP for authenticating users.

Host [required] LDAP server hostname or IP address. Can be also a comma separated list of servers to support LDAP fail-over.

Port [optional] Defaults to 389 for PLAIN un-encrypted LDAP and START_TLS. Defaults to 636 for LDAPS.

Account [optional] Only required if the LDAP server does not allow anonymous browsing of records. This should be a special account for record browsing. This will require *LDAP Password* below.

Password [optional] Only required if the LDAP server does not allow anonymous browsing of records.

Connection Security [required] Defines the connection to LDAP server

PLAIN Plain unencrypted LDAP connection. This will by default use *Port* 389.

LDAPS Use secure LDAPS connections according to *Certificate Checks* configuration. This will by default use *Port* 636.

START_TLS Use START TLS according to *Certificate Checks* configuration on an apparently "plain" LDAP connection. This will by default use *Port* 389.

Certificate Checks [optional] How SSL certificates verification is handled – this is only useful when *Enable LDAPS* is enabled. Only DEMAND or HARD offer full SSL security with mandatory certificate validation, while the other options are susceptible to man-in-the-middle attacks.

NEVER A serve certificate will never be requested or checked.

ALLOW A server certificate is requested. Failure to provide a certificate or providing a bad certificate will not terminate the session.

TRY A server certificate is requested. Failure to provide a certificate does not halt the session; providing a bad certificate halts the session.

DEMAND A server certificate is requested and must be provided and authenticated for the session to proceed.

HARD The same as DEMAND.

Custom CA Certificates [optional] Directory used by OpenSSL to find CAs for validating the LDAP server certificate. It defaults to using the system certificate store, and it should thus not be necessary to specify *Custom CA Certificates* when using certificates signed by a CA trusted by the system. It can be set to something like */etc/openldap/cacerts* on older systems or if using self-signed certificates.

Base DN [required] The Distinguished Name (DN) where searches for users will be performed. Searches can be controlled by *LDAP Filter* and *LDAP Search Scope*.

LDAP Filter [optional] A LDAP filter defined by RFC 2254. This is more useful when *LDAP Search Scope* is set to SUBTREE. The filter is useful for limiting which LDAP objects are identified as representing Users for authentication. The filter is augmented by *Login Attribute* below. This can commonly be left blank.

LDAP Search Scope [required] This limits how far LDAP will search for a matching object.

BASE Only allows searching of *Base DN* and is usually not what you want.

ONELEVEL Searches all entries under *Base DN*, but not Base DN itself.

SUBTREE Searches all entries below *Base DN*, but not Base DN itself. When using SUBTREE *LDAP Filter* is useful to limit object location.

Login Attribute [required] The LDAP record attribute that will be matched as the USERNAME or ACCOUNT used to connect to Kallithea. This will be added to *LDAP Filter* for locating the User object. If *LDAP Filter* is specified as "LDAPFILTER", *Login Attribute* is specified as "uid" and the user has connected as "jsmith" then the *LDAP Filter* will be augmented as below

```
(&(LDAPFILTER)(uid=jsmith))
```

First Name Attribute [required] The LDAP record attribute which represents the user's first name.

Last Name Attribute [required] The LDAP record attribute which represents the user's last name.

Email Attribute [required] The LDAP record attribute which represents the user's email address.

If all data are entered correctly, and python-ldap is properly installed users should be granted access to Kallithea with LDAP accounts. At this time user information is copied from LDAP into the Kallithea user database. This means that updates of an LDAP user object may not be reflected as a user update in Kallithea.

If You have problems with LDAP access and believe You entered correct information check out the Kallithea logs, any error messages sent from LDAP will be saved there.

Active Directory

Kallithea can use Microsoft Active Directory for user authentication. This is done through an LDAP or LDAPS connection to Active Directory. The following LDAP configuration settings are typical for using Active Directory

```
Base DN              = OU=SBSUsers,OU=Users,OU=MyBusiness,DC=v3sys,DC=local
Login Attribute      = sAMAccountName
First Name Attribute = givenName
Last Name Attribute  = sn
Email Attribute     = mail
```

All other LDAP settings will likely be site-specific and should be appropriately configured.

2.9.2 Authentication by container or reverse-proxy

Kallithea supports delegating the authentication of users to its WSGI container, or to a reverse-proxy server through which all clients access the application.

When these authentication methods are enabled in Kallithea, it uses the username that the container/proxy (Apache or Nginx, etc.) provides and doesn't perform the authentication itself. The authorization, however, is still done by Kallithea according to its settings.

When a user logs in for the first time using these authentication methods, a matching user account is created in Kallithea with default permissions. An administrator can then modify it using Kallithea's admin interface.

It's also possible for an administrator to create accounts and configure their permissions before the user logs in for the first time, using the *create_user* API.

Container-based authentication

In a container-based authentication setup, Kallithea reads the user name from the `REMOTE_USER` server variable provided by the WSGI container.

After setting up your container (see *Apache with mod_wsgi*), you'll need to configure it to require authentication on the location configured for Kallithea.

Proxy pass-through authentication

In a proxy pass-through authentication setup, Kallithea reads the user name from the `X-Forwarded-User` request header, which should be configured to be sent by the reverse-proxy server.

After setting up your proxy solution (see *Apache virtual host reverse proxy example*, *Apache as subdirectory* or *Nginx virtual host example*), you'll need to configure the authentication and add the username in a request header named `X-Forwarded-User`.

For example, the following config section for Apache sets a subdirectory in a reverse-proxy setup with basic auth:

```
<Location /someprefix>
  ProxyPass http://127.0.0.1:5000/someprefix
  ProxyPassReverse http://127.0.0.1:5000/someprefix
  SetEnvIf X-Url-Scheme https HTTPS=1

  AuthType Basic
  AuthName "Kallithea authentication"
  AuthUserFile /srv/kallithea/.htpasswd
```

(continues on next page)

(continued from previous page)

```
  Require valid-user

  RequestHeader unset X-Forwarded-User

  RewriteEngine On
  RewriteCond %{LA-U:REMOTE_USER} (.+)
  RewriteRule .* - [E=RU:%1]
  RequestHeader set X-Forwarded-User %{RU}e
</Location>
```

Setting metadata in container/reverse-proxy

When a new user account is created on the first login, Kallithea has no information about the user's email and full name. So you can set some additional request headers like in the example below. In this example the user is authenticated via Kerberos and an Apache mod_python fixup handler is used to get the user information from a LDAP server. But you could set the request headers however you want.

```
<Location /someprefix>
  ProxyPass http://127.0.0.1:5000/someprefix
  ProxyPassReverse http://127.0.0.1:5000/someprefix
  SetEnvIf X-Url-Scheme https HTTPS=1

  AuthName "Kerberos Login"
  AuthType Kerberos
  Krb5Keytab /etc/apache2/http.keytab
  KrbMethodK5Passwd off
  KrbVerifyKDC on
  Require valid-user

  PythonFixupHandler ldapmetadata

  RequestHeader set X_REMOTE_USER %{X_REMOTE_USER}e
  RequestHeader set X_REMOTE_EMAIL %{X_REMOTE_EMAIL}e
  RequestHeader set X_REMOTE_FIRSTNAME %{X_REMOTE_FIRSTNAME}e
  RequestHeader set X_REMOTE_LASTNAME %{X_REMOTE_LASTNAME}e
</Location>
```

```
from mod_python import apache
import ldap

LDAP_SERVER = "ldaps://server.mydomain.com:636"
LDAP_USER = ""
LDAP_PASS = ""
LDAP_ROOT = "dc=mydomain,dc=com"
LDAP_FILTER = "sAMAccountName=%s"
LDAP_ATTR_LIST = ['sAMAccountName','givenname','sn','mail']

def fixuphandler(req):
    if req.user is None:
        # no user to search for
        return apache.OK
    else:
        try:
            if('\\' in req.user):
```

(continues on next page)

(continued from previous page)

```
                username = req.user.split('\\')[1]
            elif('@' in req.user):
                username = req.user.split('@')[0]
            else:
                username = req.user
            l = ldap.initialize(LDAP_SERVER)
            l.simple_bind_s(LDAP_USER, LDAP_PASS)
            r = l.search_s(LDAP_ROOT, ldap.SCOPE_SUBTREE, LDAP_FILTER % username,␣
↪attrlist=LDAP_ATTR_LIST)

            req.subprocess_env['X_REMOTE_USER'] = username
            req.subprocess_env['X_REMOTE_EMAIL'] = r[0][1]['mail'][0].lower()
            req.subprocess_env['X_REMOTE_FIRSTNAME'] = "%s" % r[0][1]['givenname'][0]
            req.subprocess_env['X_REMOTE_LASTNAME'] = "%s" % r[0][1]['sn'][0]
        except Exception, e:
            apache.log_error("error getting data from ldap %s" % str(e), apache.APLOG_
↪ERR)

        return apache.OK
```

Note: If you enable proxy pass-through authentication, make sure your server is only accessible through the proxy. Otherwise, any client would be able to forge the authentication header and could effectively become authenticated using any account of their liking.

2.10 Version control systems setup

Kallithea supports Git and Mercurial repositories out-of-the-box. For Git, you do need the `git` command line client installed on the server.

You can always disable Git or Mercurial support by editing the file `kallithea/__init__.py` and commenting out the backend. For example, to disable Git but keep Mercurial enabled:

```
BACKENDS = {
    'hg': 'Mercurial repository',
    #'git': 'Git repository',
}
```

2.10.1 Git-specific setup

Web server with chunked encoding

Large Git pushes require an HTTP server with support for chunked encoding for POST. The Python web servers waitress and gunicorn (Linux only) can be used. By default, Kallithea uses waitress for *gearbox serve* instead of the built-in *paste* WSGI server.

The web server used by gearbox is controlled in the .ini file:

```
use = egg:waitress#main
```

or:

```
use = egg:gunicorn#main
```

Also make sure to comment out the following options:

```
threadpool_workers =
threadpool_max_requests =
use_threadpool =
```

Increasing Git HTTP POST buffer size

If Git pushes fail with HTTP error code 411 (Length Required), you may need to increase the Git HTTP POST buffer. Run the following command as the user that runs Kallithea to set a global Git variable to this effect:

```
git config --global http.postBuffer 524288000
```

2.11 Email settings

The Kallithea configuration file has several email related settings. When these contain correct values, Kallithea will send email in the situations described below. If the email configuration is not correct so that emails cannot be sent, all mails will show up in the log output.

Before any email can be sent, an SMTP server has to be configured using the configuration file setting `smtp_server`. If required for that server, specify a username (`smtp_username`) and password (`smtp_password`), a non-standard port (`smtp_port`), whether to use "SSL" when connecting (`smtp_use_ssl`) or use STARTTLS (`smtp_use_tls`), and/or specify special ESMTP "auth" features (`smtp_auth`).

For example, for sending through gmail, use:

```
smtp_server = smtp.gmail.com
smtp_username = username
smtp_password = password
smtp_port = 465
smtp_use_ssl = true
```

2.11.1 Application emails

Kallithea sends an email to *users* on several occasions:

- when comments are given on one of their changesets
- when comments are given on changesets they are reviewer on or on which they commented regardless
- when they are invited as reviewer in pull requests
- when they request a password reset

Kallithea sends an email to all *administrators* upon new account registration. Administrators are users with the `Admin` flag set on the *Admin* > *Users* page.

When Kallithea wants to send an email but due to an error cannot correctly determine the intended recipients, the administrators and the addresses specified in `email_to` in the configuration file are used as fallback.

Recipients will see these emails originating from the sender specified in the `app_email_from` setting in the configuration file. This setting can either contain only an email address, like *kallithea-noreply@example.com*, or both a name and an address in the following format: *Kallithea <kallithea-noreply@example.com>*. However, if the email is

sent due to an action of a particular user, for example when a comment is given or a pull request created, the name of that user will be combined with the email address specified in `app_email_from` to form the sender (and any name part in that configuration setting disregarded).

The subject of these emails can optionally be prefixed with the value of `email_prefix` in the configuration file.

A Kallithea-specific header indicating the email type will be added to each email. This header can be used for email filtering. The header is of the form:

    X-Kallithea-Notification-Type: <type>

where `<type>` is one of:

- `pull_request`: you are invited as reviewer in a pull request
- `pull_request_comment`: a comment was given on a pull request
- `cs_comment`: a comment was given on a changeset
- `registration`: a new user was registered
- `message`: another type of email

2.11.2 Error emails

When an exception occurs in Kallithea – and unless interactive debugging is enabled using `set debug = true` in the `[app:main]` section of the configuration file – an email with exception details is sent by backlash to the addresses specified in `email_to` in the configuration file.

Recipients will see these emails originating from the sender specified in the `error_email_from` setting in the configuration file. This setting can either contain only an email address, like *kallithea-noreply@example.com*, or both a name and an address in the following format: *Kallithea Errors <kallithea-noreply@example.com>*.

2.11.3 References

- Error Middleware (Pylons documentation)
- ErrorHandler (Pylons modules documentation)

2.12 Customization

There are several ways to customize Kallithea to your needs depending on what you want to achieve.

2.12.1 HTML/JavaScript/CSS customization

To customize the look-and-feel of the web interface (for example to add a company banner or some JavaScript widget or to tweak the CSS style definitions) you can enter HTML code (possibly with JavaScript and/or CSS) directly via the *Admin > Settings > Global > HTML/JavaScript customization block*.

2.12.2 Style sheet customization with Less

Kallithea uses Bootstrap 3 and Less for its style definitions. If you want to make some customizations, we recommend to do so by creating a `theme.less` file. When you create a file named `theme.less` in directory `kallithea/front-end/` inside the Kallithea installation, you can use this file to override the default style. For example, you

can use this to override `@kallithea-theme-main-color`, `@kallithea-logo-url` or other Bootstrap variables.

After creating the `theme.less` file, you need to regenerate the CSS files, by running:

```
kallithea-cli front-end-build --no-install-deps
```

2.12.3 Behavioral customization: Kallithea extensions

Some behavioral customization can be done in Python using Kallithea `extensions`, a custom Python file you can create to extend Kallithea functionality.

With `extensions` it's possible to add additional mappings for Whoosh indexing and statistics, to add additional code into the push/pull/create/delete repository hooks (for example to send signals to build bots such as Jenkins) and even to monkey-patch certain parts of the Kallithea source code (for example overwrite an entire function, change a global variable, ...).

To generate a skeleton extensions package, run:

```
kallithea-cli extensions-create -c my.ini
```

This will create an `extensions.py` file next to the specified `ini` file. You can find more details inside this file.

For compatibility with previous releases of Kallithea, a directory named `rcextensions` with a file `__init__.py` inside of it can also be used. If both an `extensions.py` file and an `rcextensions` directory are found, only `extensions.py` will be loaded. Note that the name `rcextensions` is deprecated and support for it will be removed in a future release.

2.12.4 Behavioral customization: code changes

As Kallithea is open-source software, you can make any changes you like directly in the source code.

We encourage you to send generic improvements back to the community so that Kallithea can become better. See *Contributing to Kallithea* for more details.

Maintenance

2.13 Backing up Kallithea

2.13.1 Settings

Just copy your .ini file, it contains all Kallithea settings.

2.13.2 Whoosh index

The Whoosh index is located in the `data/index` directory where you installed Kallithea, i.e., the same place where the ini file is located

2.13.3 Database

When using sqlite just copy kallithea.db. Any other database engine requires a manual backup operation.

A database backup will contain all gathered statistics.

2.14 Optimizing Kallithea performance

When serving a large amount of big repositories, Kallithea can start performing slower than expected. Because of the demanding nature of handling large amounts of data from version control systems, here are some tips on how to get the best performance.

2.14.1 Fast storage

Kallithea is often I/O bound, and hence a fast disk (SSD/SAN) and plenty of RAM is usually more important than a fast CPU.

2.14.2 Caching

Tweak beaker cache settings in the ini file. The actual effect of that is questionable.

Note: Beaker has no upper bound on cache size and will never drop any caches. For memory cache, the only option is to regularly restart the worker process. For file cache, it must be cleaned manually, as described in the Beaker documentation:

```
find data/cache -type f -mtime +30 -print -exec rm {} \;
```

2.14.3 Database

SQLite is a good option when having a small load on the system. But due to locking issues with SQLite, it is not recommended to use it for larger deployments.

Switching to PostgreSQL or MariaDB/MySQL will result in an immediate performance increase. A tool like SQLAlchemyGrate can be used for migrating to another database platform.

2.14.4 Horizontal scaling

Scaling horizontally means running several Kallithea instances (also known as worker processes) and let them share the load. That is essential to serve other users while processing a long-running request from a user. Usually, the bottleneck on a Kallithea server is not CPU but I/O speed - especially network speed. It is thus a good idea to run multiple worker processes on one server.

Note: Kallithea and the embedded Mercurial backend are not thread-safe. Each worker process must thus be single-threaded.

Web servers can usually launch multiple worker processes - for example `mod_wsgi` with the `WSGIDaemonProcess processes` parameter or `uWSGI` or `gunicorn` with their `workers` setting.

Kallithea can also be scaled horizontally across multiple machines. In order to scale horizontally on multiple machines, you need to do the following:

- Each instance's `data` storage needs to be configured to be stored on a shared disk storage, preferably together with repositories. This `data` dir contains template caches, sessions, whoosh index and is used for task locking (so it is safe across multiple instances). Set the `cache_dir`, `index_dir`, `beaker.cache.data_dir`, `beaker.cache.lock_dir` variables in each .ini file to a shared location across Kallithea instances
- If using several Celery instances, the message broker should be common to all of them (e.g., one shared RabbitMQ server)
- Load balance using round robin or IP hash, recommended is writing LB rules that will separate regular user traffic from automated processes like CI servers or build bots.

2.14.5 Serve static files directly from the web server

With the default `static_files` ini setting, the Kallithea WSGI application will take care of serving the static files from `kallithea/public/` at the root of the application URL.

The actual serving of the static files is very fast and unlikely to be a problem in a Kallithea setup - the responses generated by Kallithea from database and repository content will take significantly more time and resources.

To serve static files from the web server, use something like this Apache config snippet:

```
Alias /images/ /srv/kallithea/kallithea/kallithea/public/images/
Alias /css/ /srv/kallithea/kallithea/kallithea/public/css/
Alias /js/ /srv/kallithea/kallithea/kallithea/public/js/
Alias /codemirror/ /srv/kallithea/kallithea/kallithea/public/codemirror/
Alias /fontello/ /srv/kallithea/kallithea/kallithea/public/fontello/
```

Then disable serving of static files in the `.ini` `app:main` section:

```
static_files = false
```

If using Kallithea installed as a package, you should be able to find the files under `site-packages/kallithea`, either in your Python installation or in your virtualenv. When upgrading, make sure to update the web server configuration too if necessary.

It might also be possible to improve performance by configuring the web server to compress responses (served from static files or generated by Kallithea) when serving them. That might also imply buffering of responses - that is more likely to be a problem; large responses (clones or pulls) will have to be fully processed and spooled to disk or memory before the client will see any response. See the documentation for your web server.

2.15 Debugging Kallithea

If you encounter problems with Kallithea, here are some instructions on how to debug them.

Note: First make sure you're using the latest version available.

2.15.1 Enable detailed debug

Kallithea uses the standard Python `logging` module to log its output. By default only loggers with `INFO` level are displayed. To enable full output change `level = DEBUG` for all logging handlers in the currently used .ini file. This

change will allow you to see much more detailed output in the log file or console. This generally helps a lot to track issues.

2.15.2 Enable interactive debug mode

To enable interactive debug mode simply comment out `set debug = false` in the .ini file. This will trigger an interactive debugger each time there is an error in the browser, or send a http link if an error occurred in the backend. This is a great tool for fast debugging as you get a handy Python console right in the web view.

> **Warning:** NEVER ENABLE THIS ON PRODUCTION! The interactive console can be a serious security threat to your system.

2.16 Troubleshooting

Q Missing static files?

A Make sure either to set the `static_files = true` in the .ini file or double check the root path for your http setup. It should point to for example: `/home/my-virtual-python/lib/python3.7/site-packages/kallithea/public`

Q Can't install celery/rabbitmq?

A Don't worry. Kallithea works without them, too. No extra setup is required. Try out the great Celery docs for further help.

Q Long lasting push timeouts?

A Make sure you set a longer timeout in your proxy/fcgi settings. Timeouts are caused by the http server and not Kallithea.

Q Large pushes timeouts?

A Make sure you set a proper `max_body_size` for the http server. Very often Apache, Nginx, or other http servers kill the connection due to to large body.

Q Apache doesn't pass basicAuth on pull/push?

A Make sure you added `WSGIPassAuthorization true`.

Q Git fails on push/pull?

A Make sure you're using a WSGI http server that can handle chunked encoding such as `waitress` or `gunicorn`.

Q How can I use hooks in Kallithea?

A If using Mercurial, use *Admin* > *Settings* > *Hooks* to install global hooks. Inside the hooks, you can use the current working directory to control different behaviour for different repositories.

If using Git, install the hooks manually in each repository, for example by creating a file `gitrepo/hooks/pre-receive`. Note that Kallithea uses the `post-receive` hook internally. Kallithea will not work properly if another post-receive hook is installed instead. You might also accidentally overwrite your own post-receive hook with the Kallithea hook. Instead, put your post-receive hook in `post-receive-custom`, and the Kallithea hook will invoke it.

You can also use Kallithea-extensions to connect to callback hooks, for both Git and Mercurial.

Q Kallithea is slow for me, how can I make it faster?

A See the *Optimizing Kallithea performance* section.

Q UnicodeDecodeError on Apache mod_wsgi

A Please read: https://docs.djangoproject.com/en/dev/howto/deployment/wsgi/modwsgi/#if-you-get-a-unicodeencodeerror.

Q Requests hanging on Windows

A Please try out with disabled Antivirus software, there are some known problems with Eset Antivirus. Make sure you have installed the latest Windows patches (especially KB2789397).

CHAPTER 3

User guide

3.1 General Kallithea usage

3.1.1 Repository deletion

When an admin or owner deletes a repository, Kallithea does not physically delete said repository from the filesystem, but instead renames it in a special way so that it is not possible to push, clone or access the repository.

There is a special command for cleaning up such archived repositories:

```
kallithea-cli repo-purge-deleted -c my.ini --older-than=30d
```

This command scans for archived repositories that are older than 30 days, displays them, and asks if you want to delete them (unless given the `--no-ask` flag). If you host a large amount of repositories with forks that are constantly being deleted, it is recommended that you run this command via crontab.

It is worth noting that even if someone is given administrative access to Kallithea and deletes a repository, you can easily restore such an action by renaming the repository directory, removing the `rm__<date>` prefix.

3.1.2 File view: follow current branch

In the file view, left and right arrows allow to jump to the previous and next revision. Depending on the way revisions were created in the repository, this could jump to a different branch. When the checkbox `Follow current branch` is checked, these arrows will only jump to revisions on the same branch as the currently visible revision. So for example, if someone is viewing files in the `beta` branch and marks the *Follow current branch* checkbox, the < and > arrows will only show revisions on the `beta` branch.

3.1.3 Changelog features

The core feature of a repository's `changelog` page is to show the revisions in a repository. However, there are several other features available from the changelog.

Branch filter By default, the changelog shows revisions from all branches in the repository. Use the branch filter to restrict to a given branch.

Viewing a changeset A particular changeset can be opened by clicking on either the changeset hash or the commit message, or by ticking the checkbox and clicking the `Show selected changeset` button at the top.

Viewing all changes between two changesets To get a list of all changesets between two selected changesets, along with the changes in each one of them, tick the checkboxes of the first and last changeset in the desired range and click the `Show selected changesets` button at the top. You can only show the range between the first and last checkbox (no cherry-picking).

From that page, you can proceed to viewing the overall delta between the selected changesets, by clicking the `Compare revisions` button.

Creating a pull request You can create a new pull request for the changes of a particular changeset (and its ancestors) by selecting it and clicking the `Open new pull request for selected changesets` button.

3.1.4 Permanent repository URLs

Due to the complicated nature of repository grouping, URLs of repositories can often change. For example, a repository originally accessible from:

```
http://kallithea.example.com/repo_name
```

would get a new URL after moving it to test_group:

```
http://kallithea.example.com/test_group/repo_name
```

Such moving of a repository to a group can be an issue for build systems and other scripts where the repository paths are hardcoded. To mitigate this, Kallithea provides permanent URLs using the repository ID prefixed with an underscore. In all Kallithea URLs, for example those for the changelog and the file view, a repository name can be replaced by this `_ID` string. Since IDs are always the same, moving the repository to a different group will not affect such URLs.

In the example, the repository could also be accessible as:

```
http://kallithea.example.com/_<ID>
```

The ID of a given repository can be shown from the repository `Summary` page, by selecting the `Show by ID` button next to `Clone URL`.

3.1.5 Email notifications

With email settings properly configured in the Kallithea configuration file, Kallithea will send emails on user registration and when errors occur.

Emails are also sent for comments on changesets. In this case, an email is sent to the committer of the changeset (if known to Kallithea), to all reviewers of the pull request (if applicable) and to all people mentioned in the comment using @mention notation.

3.1.6 Trending source files

Trending source files are calculated based on a predefined dictionary of known types and extensions. If an extension is missing or you would like to scan custom files, it is possible to add additional file extensions with `EXTRA_MAPPINGS` in your custom Kallithea extensions.py file. See *Customization*.

3.1.7 Cloning remote repositories

Kallithea has the ability to clone repositories from given remote locations.

If you need to clone repositories that are protected via basic authentication, you can pass the credentials in the URL, e.g. `http://user:passw@remote.example.com/repo`. Kallithea will then try to login and clone using the given credentials. Please note that the given credentials will be stored as plaintext inside the database. However, the authentication information will not be shown in the clone URL on the summary page.

3.1.8 Specific features configurable in the Admin settings

In general, the Admin settings should be self-explanatory and will not be described in more detail in this documentation. However, there are a few features that merit further explanation.

Repository extra fields

In the *Visual* tab, there is an option "Use repository extra fields", which allows to set custom fields for each repository in the system.

Once enabled site-wide, the custom fields can be edited per-repository under *Options* | *Settings* | *Extra Fields*.

Example usage of such fields would be to define company-specific information into repositories, e.g., defining a `repo_manager` key that would give info about a manager of each repository. There's no limit for adding custom fields. Newly created fields are accessible via the API.

Meta tagging

In the *Visual* tab, option "Stylify recognised meta tags" will cause Kallithea to turn certain text fragments in repository and repository group descriptions into colored tags. Currently recognised tags are:

```
[featured]
[stale]
[dead]
[lang => lang]
[license => License]
[requires => Repo]
[recommends => Repo]
[see => URI]
```

3.2 Version control systems usage notes

3.2.1 Importing existing repositories

There are two main methods to import repositories in Kallithea: via the web interface or via the filesystem. If you have a large number of repositories to import, importing them via the filesystem is more convenient.

Importing via web interface

For a small number of repositories, it may be easier to create the target repositories through the Kallithea web interface, via *Admin* > *Repositories* or via the *Add Repository* button on the entry page of the web interface.

Repositories can be nested in repository groups by first creating the group (via *Admin* > *Repository Groups* or via the *Add Repository Group* button on the entry page of the web interface) and then selecting the appropriate group when adding the repository.

After creation of the (empty) repository, push the existing commits to the *Clone URL* displayed on the repository summary page. For Git repositories, first add the *Clone URL* as remote, then push the commits to that remote. The specific commands to execute are shown under the *Existing repository?* section of the new repository's summary page.

A benefit of this method particular for Git repositories, is that the Kallithea-specific Git hooks are installed automatically. For Mercurial, no hooks are required anyway.

Importing via the filesystem

The alternative method of importing repositories consists of creating the repositories in the desired hierarchy on the filesystem and letting Kallithea scan that location.

All repositories are stored in a central location on the filesystem. This location is specified during installation (via `db-create`) and can be reviewed at *Admin* > *Settings* > *VCS* > *Location of repositories*. Repository groups (defined in *Admin* > *Repository Groups*) are represented by a directory in that repository location. Repositories of the repository group are nested under that directory.

To import a set of repositories and organize them in a certain repository group structure, first place clones in the desired hierarchy at the configured repository location. These clones should be created without working directory. For Mercurial, this is done with `hg clone -U`, for Git with `git clone --bare`.

When the repositories are added correctly on the filesystem:

- go to *Admin* > *Settings* > *Remap and Rescan* in the Kallithea web interface
- select the *Install Git hooks* checkbox when importing Git repositories
- click *Rescan Repositories*

This step will scan the filesystem and create the appropriate repository groups and repositories in Kallithea.

Note: Once repository groups have been created this way, manage their access permissions through the Kallithea web interface.

3.2.2 Mercurial-specific notes

Working with subrepositories

This section explains how to use Mercurial subrepositories in Kallithea.

Example usage:

```
## init a simple repo
hg init mainrepo
cd mainrepo
echo "file" > file
hg add file
hg ci --message "initial file"

# clone subrepo we want to add from Kallithea
hg clone http://kallithea.local/subrepo

## specify URL to existing repo in Kallithea as subrepository path
echo "subrepo = http://kallithea.local/subrepo" > .hgsub
```

(continues on next page)

(continued from previous page)

```
hg add .hgsub
hg ci --message "added remote subrepo"
```

In the file list of a clone of `mainrepo` you will see a connected subrepository at the revision it was cloned with. Clicking on the subrepository link sends you to the proper repository in Kallithea.

Cloning `mainrepo` will also clone the attached subrepository.

Next we can edit the subrepository data, and push back to Kallithea. This will update both repositories.

3.3 Repository statistics

Kallithea has a *repository statistics* feature, disabled by default. When enabled, the amount of commits per committer is visualized in a timeline. This feature can be enabled using the `Enable statistics` checkbox on the repository `Settings` page.

The statistics system makes heavy demands on the server resources, so in order to keep a balance between usability and performance, statistics are cached inside the database and gathered incrementally.

When Celery is disabled:

> On each first visit to the summary page a set of 250 commits are parsed and added to the statistics cache. This incremental gathering also happens on each visit to the statistics page, until all commits are fetched.
>
> Statistics are kept cached until additional commits are added to the repository. In such a case Kallithea will only fetch the new commits when updating its statistics cache.

When Celery is enabled:

> On the first visit to the summary page, Kallithea will create tasks that will execute on Celery workers. These tasks will gather all of the statistics until all commits are parsed. Each task parses 250 commits, then launches a new task.

3.4 API

Kallithea has a simple JSON RPC API with a single schema for calling all API methods. Everything is available by sending JSON encoded http(s) requests to `<your_server>/_admin/api`.

3.4.1 API keys

Every Kallithea user automatically receives an API key, which they can view under “My Account”. On this page, API keys can also be revoked, and additional API keys can be generated.

3.4.2 API access

Clients must send JSON encoded JSON-RPC requests:

```
{
    "id": "<id>",
    "api_key": "<api_key>",
    "method": "<method_name>",
```

(continues on next page)

(continued from previous page)

```
    "args": {"<arg_key>": "<arg_val>"}
}
```

For example, to pull to a local "CPython" mirror using curl:

```
curl https://kallithea.example.com/_admin/api -X POST -H 'content-type:text/plain' \
    --data-binary '{"id":1,"api_key":"xe7cdb2v278e4evbdf5vs04v832v0efvcbcve4a3",
↪"method":"pull","args":{"repoid":"CPython"}}'
```

In general, provide

- *id*, a value of any type, can be used to match the response with the request that it is replying to.
- *api_key*, for authentication and permission validation.
- *method*, the name of the method to call – a list of available methods can be found below.
- *args*, the arguments to pass to the method.

Note: api_key can be found or set on the user account page.

The response to the JSON-RPC API call will always be a JSON structure:

```
{
    "id": <id>,  # the id that was used in the request
    "result": <result>|null,  # JSON formatted result (null on error)
    "error": null|<error_message>  # JSON formatted error (null on success)
}
```

All responses from the API will be `HTTP/1.0 200 OK`. If an error occurs, the reponse will have a failure description in *error* and *result* will be null.

3.4.3 API client

Kallithea comes with a `kallithea-api` command line tool, providing a convenient way to call the JSON-RPC API.

For example, to call `get_repo`:

```
kallithea-api --apihost=<Kallithea URL> --apikey=<API key> get_repo

Calling method get_repo => <Kallithea URL>
Server response
ERROR:"Missing non optional `repoid` arg in JSON DATA"
```

Oops, looks like we forgot to add an argument. Let's try again, now providing the `repoid` as a parameter:

```
kallithea-api --apihost=<Kallithea URL> --apikey=<API key> get_repo repoid:myrepo

Calling method get_repo => <Kallithea URL>
Server response
{
    "clone_uri": null,
    "created_on": "2015-08-31T14:55:19.042",
...
```

To avoid specifying `apihost` and `apikey` every time, run:

```
kallithea-api --save-config --apihost=<Kallithea URL> --apikey=<API key>
```

This will create a `~/.config/kallithea` with the specified URL and API key so you don't have to specify them every time.

3.4.4 API methods

pull

Pull the given repo from remote location. Can be used to automatically keep remote repos up to date. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "pull"
args :    {
            "repoid" : "<reponame or repo_id>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : "Pulled from `<reponame>`"
error : null
```

rescan_repos

Rescan repositories. If `remove_obsolete` is set, Kallithea will delete repos that are in the database but not in the filesystem. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "rescan_repos"
args :    {
            "remove_obsolete" : "<boolean = Optional(False)>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : "{'added': [<list of names of added repos>],
           'removed': [<list of names of removed repos>]}"
error : null
```

invalidate_cache

Invalidate the cache for a repository. This command can only be executed using the api_key of a user with admin rights, or that of a regular user with admin or write access to the repository.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "invalidate_cache"
args :    {
            "repoid" : "<reponame or repo_id>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : "Caches of repository `<reponame>`"
error : null
```

get_ip

Return IP address as seen from Kallithea server, together with all defined IP addresses for given user. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_ip"
args :    {
            "userid" : "<user_id or username>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
             "ip_addr_server" : <ip_from_client>",
             "user_ips" : [
                             {
                                "ip_addr" : "<ip_with_mask>",
                                "ip_range" : ["<start_ip>", "<end_ip>"]
                             },
                             ...
                           ]
         }
error : null
```

get_user

Get a user by username or userid. The result is empty if user can't be found. If userid param is skipped, it is set to id of user who is calling this method. Any userid can be specified when the command is executed using the api_key of a user with admin rights. Regular users can only specify their own userid.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_user"
```

(continues on next page)

(continued from previous page)

```
args :    {
            "userid" : "<username or user_id Optional(=apiuser)>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : None if user does not exist or
         {
            "user_id" :     "<user_id>",
            "api_key" :     "<api_key>",
            "username" :    "<username>",
            "firstname" :   "<firstname>",
            "lastname" :    "<lastname>",
            "email" :       "<email>",
            "emails" :      "<list_of_all_additional_emails>",
            "ip_addresses": "<list_of_ip_addresses_for_user>",
            "active" :      "<bool>",
            "admin" :       "<bool>",
            "ldap_dn" :     "<ldap_dn>",
            "last_login" :  "<last_login>",
            "permissions": {
                "global": ["hg.create.repository",
                           "repository.read",
                           "hg.register.manual_activate"],
                "repositories" : {"repo1" : "repository.none"},
                "repositories_groups" : {"Group1" : "group.read"}
             }
         }
error : null
```

get_users

List all existing users. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_users"
args :    { }
```

OUTPUT:

```
id : <id_given_in_input>
result : [
          {
            "user_id" :     "<user_id>",
            "api_key" :     "<api_key>",
            "username" :    "<username>",
            "firstname" :   "<firstname>",
            "lastname" :    "<lastname>",
            "email" :       "<email>",
            "emails" :      "<list_of_all_additional_emails>",
            "ip_addresses": "<list_of_ip_addresses_for_user>",
```

(continues on next page)

(continued from previous page)

```
            "active" :      "<bool>",
            "admin" :       "<bool>",
            "ldap_dn" :     "<ldap_dn>",
            "last_login" :  "<last_login>"
          },
          ...
        ]
error : null
```

create_user

Create new user. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "create_user"
args :    {
            "username" :  "<username>",
            "email" :     "<useremail>",
            "password" :  "<password = Optional(None)>",
            "firstname" : "<firstname> = Optional(None)",
            "lastname" :  "<lastname> = Optional(None)",
            "active" :    "<bool> = Optional(True)",
            "admin" :     "<bool> = Optional(False)",
            "ldap_dn" :   "<ldap_dn> = Optional(None)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "created new user `<username>`",
          "user" : {
            "user_id" :  "<user_id>",
            "username" : "<username>",
            "firstname": "<firstname>",
            "lastname" : "<lastname>",
            "email" :    "<email>",
            "emails" :   "<list_of_all_additional_emails>",
            "active" :   "<bool>",
            "admin" :    "<bool>",
            "ldap_dn" :  "<ldap_dn>",
            "last_login": "<last_login>"
          }
         }
error : null
```

Example:

```
kallithea-api create_user username:bent email:bent@example.com firstname:Bent␣
↪lastname:Bentsen extern_type:ldap extern_name:uid=bent,dc=example,dc=com
```

update_user

Update the given user if such user exists. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "update_user"
args :    {
            "userid" : "<user_id or username>",
            "username" :  "<username> = Optional(None)",
            "email" :     "<useremail> = Optional(None)",
            "password" :  "<password> = Optional(None)",
            "firstname" : "<firstname> = Optional(None)",
            "lastname" :  "<lastname> = Optional(None)",
            "active" :    "<bool> = Optional(None)",
            "admin" :     "<bool> = Optional(None)",
            "ldap_dn" :   "<ldap_dn> = Optional(None)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "updated user ID:<userid> <username>",
          "user" : {
            "user_id" :  "<user_id>",
            "api_key" :  "<api_key>",
            "username" : "<username>",
            "firstname": "<firstname>",
            "lastname" : "<lastname>",
            "email" :    "<email>",
            "emails" :   "<list_of_all_additional_emails>",
            "active" :   "<bool>",
            "admin" :    "<bool>",
            "ldap_dn" :  "<ldap_dn>",
            "last_login": "<last_login>"
          }
         }
error : null
```

delete_user

Delete the given user if such a user exists. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "delete_user"
args :    {
            "userid" : "<user_id or username>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "deleted user ID:<userid> <username>",
          "user" : null
         }
error : null
```

get_user_group

Get an existing user group. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_user_group"
args :    {
            "usergroupid" : "<user group id or name>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : None if group not exist
         {
           "users_group_id" : "<id>",
           "group_name" :     "<groupname>",
           "active" :         "<bool>",
           "members" :  [
                          {
                            "user_id" :  "<user_id>",
                            "api_key" :  "<api_key>",
                            "username" : "<username>",
                            "firstname": "<firstname>",
                            "lastname" : "<lastname>",
                            "email" :    "<email>",
                            "emails" :   "<list_of_all_additional_emails>",
                            "active" :   "<bool>",
                            "admin" :    "<bool>",
                            "ldap_dn" :  "<ldap_dn>",
                            "last_login": "<last_login>"
                          },
                          ...
                        ]
         }
error : null
```

get_user_groups

List all existing user groups. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
```

(continues on next page)

(continued from previous page)

```
method :  "get_user_groups"
args :    { }
```

OUTPUT:

```
id : <id_given_in_input>
result : [
           {
           "users_group_id" : "<id>",
           "group_name" :     "<groupname>",
           "active" :         "<bool>"
           },
           ...
          ]
error : null
```

create_user_group

Create a new user group. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "create_user_group"
args :    {
            "group_name": "<groupname>",
            "owner" :     "<owner_name_or_id = Optional(=apiuser)>",
            "active" :    "<bool> = Optional(True)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "created new user group `<groupname>`",
          "users_group" : {
                 "users_group_id" : "<id>",
                 "group_name" :     "<groupname>",
                 "active" :         "<bool>"
           }
         }
error : null
```

add_user_to_user_group

Add a user to a user group. If the user already is in that group, success will be `false`. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "add_user_user_group"
```

(continues on next page)

(continued from previous page)

```
args :    {
            "usersgroupid" : "<user group id or name>",
            "userid" : "<user_id or username>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "success" : True|False,  # depends on if member is in group
          "msg" : "added member `<username>` to a user group `<groupname>` |
                   User is already in that group"
         }
error : null
```

remove_user_from_user_group

Remove a user from a user group. If the user isn't in the given group, success will be `false`. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "remove_user_from_user_group"
args :    {
            "usersgroupid" : "<user group id or name>",
            "userid" : "<user_id or username>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "success" : True|False,  # depends on if member is in group
          "msg" : "removed member <username> from user group <groupname> |
                   User wasn't in group"
         }
error : null
```

get_repo_group

Get an existing repository group. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_repo_group"
args :    {
            "repogroupid" : "<repo group id or name>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result :
           {
           "group_id" :           "<id>",
           "group_name" :         "<groupname>",
           "group_description" : "<groupdescription>",
           "parent_group" :       "<groupid>"|null,
           "repositories" :       "<list_of_all_repo_names_in_group>",
           "owner" :              "<owner>",
           "members" :            [
                                    {
                                      "name" : "<name>",
                                      "type" : "user",
                                      "permission" : "group.(none|read|write|admin)"
                                    },
                                    {
                                      "name" : "<name>",
                                      "type" : "user_group",
                                      "permission" : "group.(none|read|write|admin)"
                                    },
                                    ...
                                  ]

           },
error : null
```

get_repo_groups

List all existing repository groups. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_repo_groups"
args :    { }
```

OUTPUT:

```
id : <id_given_in_input>
result : [
           {
           "group_id" :           "<id>",
           "group_name" :         "<groupname>",
           "group_description" : "<groupdescription>",
           "parent_group" :       "<groupid>"|null,
           "repositories" :       "<list_of_all_repo_names_in_group>",
           "owner" :              "<owner>"
           },
           ...
         ]
error : null
```

create_repo_group

Create a new repository group. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "create_repo_group"
args :    {
            "group_name" :       "<group_name>",
            "description" :      "<description> = Optional("")",
            "owner" :            "<username or user_id> = Optional(None)",
            "parent" :           "<reponame or id> = Optional(None)",
            "copy_permissions" : "<bool> = Optional(False)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
            "msg" : "created new repo group `<group_name>`",
            "repo_group" : {
                             "group_id" :          <id>,
                             "group_name" :        "<parent_group>/<group_name>",
                             "group_description" : "<description>",
                             "parent_group" :      <id>|null,
                             "repositories" :      <list of repositories>,
                             "owner" :             "<user_name>"
                           }
```

update_repo_group

Update a repository group. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "update_repo_group"
args :    {
            "repogroupid" :         "<id>",
            "group_name" :       "<group_name> = Optional(None)",
            "description" :      "<description> = Optional(None)",
            "owner" :            "<username or user_id> = Optional(None)",
            "parent" :           "<reponame or id> = Optional(None)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
            "msg" : "updated repository group ID:<id> <group_name>",
            "repo_group" : {
                             "group_id" :          <id>,
                             "group_name" :        "<parent_group>/<group_name>",
                             "group_description" : "<description>",
                             "parent_group" :      <id>|null,
                             "repositories" :      <list of repositories>,
                             "owner" :             "<user_name>"
                           }
```

delete_repo_group

Delete a repository group. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "delete_repo_group"
args :    {
            "repogroupid" : "<id>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
             "msg" : "deleted repo group ID:<id> <group_name>",
             "repo_group" : null
          }
```

get_repo

Get an existing repository by its name or repository_id. Members will contain either users_group or users associated to that repository. This command can only be executed using the api_key of a user with admin rights, or that of a regular user with at least read access to the repository.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_repo"
args :    {
            "repoid" : "<reponame or repo_id>",
            "with_revision_names" : "<bool> = Optional(False)",
            "with_pullrequests" : "<bool> = Optional(False)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : None if repository does not exist or
         {
            "repo_id" :          "<repo_id>",
            "repo_name" :        "<reponame>",
            "repo_type" :        "<repo_type>",
            "clone_uri" :        "<clone_uri>",
            "enable_downloads" : "<bool>",
            "enable_statistics": "<bool>",
            "private" :          "<bool>",
            "created_on" :       "<date_time_created>",
            "description" :      "<description>",
            "landing_rev" :      "<landing_rev>",
            "last_changeset" :   {
                                     "author" :  "<full_author>",
                                     "date" :    "<date_time_of_commit>",
                                     "message" : "<commit_message>",
```

(continues on next page)

(continued from previous page)

```
                                "raw_id" :  "<raw_id>",
                                "revision": "<numeric_revision>",
                                "short_id": "<short_id>"
                            },
        "owner" :           "<repo_owner>",
        "fork_of" :         "<name_of_fork_parent>",
        "members" :   [
                            {
                              "type" :       "user",
                              "user_id" :    "<user_id>",
                              "api_key" :    "<api_key>",
                              "username" :   "<username>",
                              "firstname" :  "<firstname>",
                              "lastname" :   "<lastname>",
                              "email" :      "<email>",
                              "emails" :     "<list_of_all_additional_emails>",
                              "active" :     "<bool>",
                              "admin" :      "<bool>",
                              "ldap_dn" :    "<ldap_dn>",
                              "last_login" : "<last_login>",
                              "permission" : "repository.(read|write|admin)"
                            },
                            ...
                            {
                              "type" :     "users_group",
                              "id" :       "<usersgroupid>",
                              "name" :     "<usersgroupname>",
                              "active" :   "<bool>",
                              "permission" : "repository.(read|write|admin)"
                            },
                            ...
                        ],
        "followers" : [
                            {
                              "user_id" :     "<user_id>",
                              "username" :    "<username>",
                              "api_key" :     "<api_key>",
                              "firstname" :   "<firstname>",
                              "lastname" :    "<lastname>",
                              "email" :       "<email>",
                              "emails" :      "<list_of_all_additional_emails>",
                              "ip_addresses": "<list_of_ip_addresses_for_user>",
                              "active" :      "<bool>",
                              "admin" :       "<bool>",
                              "ldap_dn" :     "<ldap_dn>",
                              "last_login" :  "<last_login>"
                            },
                            ...
                        ],
        <if with_revision_names == True>
        "tags" : {
                    "<tagname>" : "<raw_id>",
                    ...
                },
        "branches" : {
                    "<branchname>" : "<raw_id>",
                    ...
```

(continues on next page)

(continued from previous page)

```
                },
            "bookmarks" : {
                    "<bookmarkname>" : "<raw_id>",
                    ...
                },
            <if with_pullrequests == True>
            "pull_requests" : [
              {
                "status" : "<pull_request_status>",
                "pull_request_id" : <pull_request_id>,
                "description" : "<pull_request_description>",
                "title" : "<pull_request_title>",
                "url" : "<pull_request_url>",
                "reviewers" : [
                  {
                    "username" : "<user_id>"
                  },
                  ...
                ],
                "org_repo_url" : "<repo_url>",
                "org_ref_parts" : [
                  "<ref_type>",
                  "<ref_name>",
                  "<raw_id>"
                ],
                "other_ref_parts" : [
                  "<ref_type>",
                  "<ref_name>",
                  "<raw_id>"
                ],
                "comments" : [
                  {
                    "username" : "<user_id>",
                    "text" : "<comment text>",
                    "comment_id" : "<comment_id>"
                  },
                  ...
                ],
                "owner" : "<username>",
                "statuses" : [
                  {
                    "status" : "<status_of_review>",        # "under_review",
↪"approved" or "rejected"
                    "reviewer" : "<user_id>",
                    "modified_at" : "<date_time_of_review>" # iso 8601 date, server's␣
↪timezone
                  },
                  ...
                ],
                "revisions" : [
                  "<raw_id>",
                  ...
                ]
              },
              ...
            ]
          }
```

(continues on next page)

(continued from previous page)

```
error : null
```

get_repos

List all existing repositories. This command can only be executed using the api_key of a user with admin rights, or that of a regular user with at least read access to the repository.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_repos"
args :    { }
```

OUTPUT:

```
id : <id_given_in_input>
result : [
          {
            "repo_id" :          "<repo_id>",
            "repo_name" :        "<reponame>",
            "repo_type" :        "<repo_type>",
            "clone_uri" :        "<clone_uri>",
            "private" :          "<bool>",
            "created_on" :       "<datetimecreated>",
            "description" :      "<description>",
            "landing_rev" :      "<landing_rev>",
            "owner" :            "<repo_owner>",
            "fork_of" :          "<name_of_fork_parent>",
            "enable_downloads" : "<bool>",
            "enable_statistics": "<bool>"
          },
          ...
         ]
error : null
```

get_repo_nodes

Return a list of files and directories for a given path at the given revision. It is possible to specify ret_type to show only `files` or `dirs`. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_repo_nodes"
args :    {
            "repoid" : "<reponame or repo_id>",
            "revision" :  "<revision>",
            "root_path" : "<root_path>",
            "ret_type" :  "<ret_type> = Optional('all')"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : [
          {
            "name" :         "<name>",
            "type" :         "<type>"
          },
          ...
         ]
error : null
```

create_repo

Create a repository. If the repository name contains "/", the repository will be created in the repository group indicated by that path. Any such repository groups need to exist before calling this method, or the call will fail. For example "foo/bar/baz" will create a repository "baz" inside the repository group "bar" which itself is in a repository group "foo", but both "foo" and "bar" already need to exist before calling this method. This command can only be executed using the api_key of a user with admin rights, or that of a regular user with create repository permission. Regular users cannot specify owner parameter.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "create_repo"
args :    {
            "repo_name" :         "<reponame>",
            "owner" :             "<owner_name_or_id = Optional(=apiuser)>",
            "repo_type" :         "<repo_type> = Optional('hg')",
            "description" :       "<description> = Optional('')",
            "private" :           "<bool> = Optional(False)",
            "clone_uri" :         "<clone_uri> = Optional(None)",
            "landing_rev" :       "<landing_rev> = Optional('tip')",
            "enable_downloads" :  "<bool> = Optional(False)",
            "enable_statistics": "<bool> = Optional(False)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "Created new repository `<reponame>`",
          "repo" : {
            "repo_id" :           "<repo_id>",
            "repo_name" :         "<reponame>",
            "repo_type" :         "<repo_type>",
            "clone_uri" :         "<clone_uri>",
            "private" :           "<bool>",
            "created_on" :        "<datetimecreated>",
            "description" :       "<description>",
            "landing_rev" :       "<landing_rev>",
            "owner" :             "<username or user_id>",
            "fork_of" :           "<name_of_fork_parent>",
            "enable_downloads" :  "<bool>",
            "enable_statistics": "<bool>"
          }
```

(continues on next page)

(continued from previous page)

```
          }
error : null
```

update_repo

Update a repository. This command can only be executed using the api_key of a user with admin rights, or that of a regular user with create repository permission. Regular users cannot specify owner parameter.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "update_repo"
args :    {
            "repoid" :           "<reponame or repo_id>",
            "name" :             "<reponame> = Optional('')",
            "group" :            "<group_id> = Optional(None)",
            "owner" :            "<owner_name_or_id = Optional(=apiuser)>",
            "description" :      "<description> = Optional('')",
            "private" :          "<bool> = Optional(False)",
            "clone_uri" :        "<clone_uri> = Optional(None)",
            "landing_rev" :      "<landing_rev> = Optional('tip')",
            "enable_downloads" : "<bool> = Optional(False)",
            "enable_statistics": "<bool> = Optional(False)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "updated repo ID:repo_id `<reponame>`",
          "repository" : {
            "repo_id" :          "<repo_id>",
            "repo_name" :        "<reponame>",
            "repo_type" :        "<repo_type>",
            "clone_uri" :        "<clone_uri>",
            "private" :          "<bool>",
            "created_on" :       "<datetimecreated>",
            "description" :      "<description>",
            "landing_rev" :      "<landing_rev>",
            "owner" :            "<username or user_id>",
            "fork_of" :          "<name_of_fork_parent>",
            "enable_downloads" : "<bool>",
            "enable_statistics": "<bool>",
            "last_changeset" :   {
                                   "author" :  "<full_author>",
                                   "date" :    "<date_time_of_commit>",
                                   "message" : "<commit_message>",
                                   "raw_id" :  "<raw_id>",
                                   "revision": "<numeric_revision>",
                                   "short_id": "<short_id>"
                                 }
          }
         }
error : null
```

fork_repo

Create a fork of the given repo. If using Celery, this will return success message immediately and a fork will be created asynchronously. This command can only be executed using the api_key of a user with admin rights, or with the global fork permission, by a regular user with create repository permission and at least read access to the repository. Regular users cannot specify owner parameter.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "fork_repo"
args :    {
            "repoid" :          "<reponame or repo_id>",
            "fork_name" :       "<forkname>",
            "owner" :           "<username or user_id = Optional(=apiuser)>",
            "description" :     "<description>",
            "copy_permissions": "<bool>",
            "private" :         "<bool>",
            "landing_rev" :     "<landing_rev>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "Created fork of `<reponame>` as `<forkname>`",
          "success" : true
         }
error : null
```

delete_repo

Delete a repository. This command can only be executed using the api_key of a user with admin rights, or that of a regular user with admin access to the repository. When `forks` param is set it is possible to detach or delete forks of the deleted repository.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "delete_repo"
args :    {
            "repoid" : "<reponame or repo_id>",
            "forks" :  "`delete` or `detach` = Optional(None)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "Deleted repository `<reponame>`",
          "success" : true
         }
error : null
```

grant_user_permission

Grant permission for a user on the given repository, or update the existing one if found. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "grant_user_permission"
args :    {
            "repoid" : "<reponame or repo_id>",
            "userid" : "<username or user_id>",
            "perm" :       "(repository.(none|read|write|admin))"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "Granted perm: `<perm>` for user: `<username>` in repo: `<reponame>`
↪",
          "success" : true
         }
error : null
```

revoke_user_permission

Revoke permission for a user on the given repository. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "revoke_user_permission"
args :    {
            "repoid" : "<reponame or repo_id>",
            "userid" : "<username or user_id>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "Revoked perm for user: `<username>` in repo: `<reponame>`",
          "success" : true
         }
error : null
```

grant_user_group_permission

Grant permission for a user group on the given repository, or update the existing one if found. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "grant_user_group_permission"
args :    {
            "repoid" : "<reponame or repo_id>",
            "usersgroupid" : "<user group id or name>",
            "perm" : "(repository.(none|read|write|admin))"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "Granted perm: `<perm>` for group: `<usersgroupname>` in repo: `
↪<reponame>`",
          "success" : true
         }
error : null
```

revoke_user_group_permission

Revoke permission for a user group on the given repository. This command can only be executed using the api_key of a user with admin rights.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "revoke_user_group_permission"
args :    {
            "repoid" : "<reponame or repo_id>",
            "usersgroupid" : "<user group id or name>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "msg" : "Revoked perm for group: `<usersgroupname>` in repo: `<reponame>`",
          "success" : true
         }
error : null
```

get_changesets

Get changesets of a given repository. This command can only be executed using the api_key of a user with read permissions to the repository.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_changesets"
args :    {
            "repoid" : "<reponame or repo_id>",
```

(continues on next page)

(continued from previous page)

```
            "start" : "<revision number> = Optional(None)",
            "end" : "<revision number> = Optional(None)",
            "start_date" : "<date> = Optional(None)",     # in "%Y-%m-%dT%H:%M:%S"␣
↪format
            "end_date" : "<date> = Optional(None)",       # in "%Y-%m-%dT%H:%M:%S"␣
↪format
            "branch_name" : "<branch name filter> = Optional(None)",
            "reverse" : "<bool> = Optional(False)",
            "with_file_list" : "<bool> = Optional(False)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : [
{
  "raw_id" : "<raw_id>",
  "short_id" : "<short_id>",
  "author" : "<full_author>",
  "date" : "<date_time_of_commit>",
  "message" : "<commit_message>",
  "revision" : "<numeric_revision>",
  <if with_file_list == True>
  "added" : [<list of added files>],
  "changed" : [<list of changed files>],
  "removed" : [<list of removed files>]
},
...
]
error : null
```

get_changeset

Get information and review status for a given changeset. This command can only be executed using the api_key of a user with read permissions to the repository.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_changeset"
args :    {
            "repoid" : "<reponame or repo_id>",
            "raw_id" : "<raw_id>",
            "with_reviews" : "<bool> = Optional(False)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
          "author" :  "<full_author>",
          "date" :    "<date_time_of_commit>",
          "message" : "<commit_message>",
          "raw_id" :  "<raw_id>",
```

(continues on next page)

(continued from previous page)

```
          "revision": "<numeric_revision>",
          "short_id": "<short_id>",
          "reviews" : [{
                "reviewer" :  "<username>",
                "modified_at" : "<date_time_of_review>",  # iso 8601 date, server's␣
↪timezone
                "status" :  "<status_of_review>",        # "under_review", "approved"␣
↪or "rejected"
             },
             ...
          ]
        }
error : null
```

Example output:

```
{
  "id" : 1,
  "error" : null,
  "result" : {
    "author" : {
      "email" : "user@example.com",
      "name" : "Kallithea Admin"
    },
    "changed" : [],
    "short_id" : "e1022d3d28df",
    "date" : "2017-03-28T09:09:03",
    "added" : [
      "README.rst"
    ],
    "removed" : [],
    "revision" : 0,
    "raw_id" : "e1022d3d28dfba02f626cde65dbe08f4ceb0e4e7",
    "message" : "Added file via Kallithea",
    "id" : "e1022d3d28dfba02f626cde65dbe08f4ceb0e4e7",
    "reviews" : [
      {
        "status" : "under_review",
        "modified_at" : "2017-03-28T09:17:08.618",
        "reviewer" : "user"
      }
    ]
  }
}
```

get_pullrequest

Get information and review status for a given pull request. This command can only be executed using the api_key of a user with read permissions to the original repository.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "get_pullrequest"
```

(continues on next page)

(continued from previous page)

```
args :    {
            "pullrequest_id" : "<pullrequest_id>"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : {
    "status" : "<pull_request_status>",
    "pull_request_id" : <pull_request_id>,
    "description" : "<pull_request_description>",
    "title" : "<pull_request_title>",
    "url" : "<pull_request_url>",
    "reviewers" : [
      {
        "username" : "<user_name>"
      },
      ...
    ],
    "org_repo_url" : "<repo_url>",
    "org_ref_parts" : [
      "<ref_type>",
      "<ref_name>",
      "<raw_id>"
    ],
    "other_ref_parts" : [
      "<ref_type>",
      "<ref_name>",
      "<raw_id>"
    ],
    "comments" : [
      {
        "username" : "<user_name>",
        "text" : "<comment text>",
        "comment_id" : "<comment_id>"
      },
      ...
    ],
    "owner" : "<username>",
    "statuses" : [
      {
        "status" : "<status_of_review>",        # "under_review", "approved" or
↪"rejected"
        "reviewer" : "<user_name>",
        "modified_at" : "<date_time_of_review>" # iso 8601 date, server's timezone
      },
      ...
    ],
    "revisions" : [
      "<raw_id>",
      ...
    ]
},
error : null
```

comment_pullrequest

Add comment, change status or close a given pull request. This command can only be executed using the api_key of a user with read permissions to the original repository.

INPUT:

```
id : <id_for_response>
api_key : "<api_key>"
method :  "comment_pullrequest"
args :    {
            "pull_request_id" : "<pull_request_id>",
            "comment_msg" :     Optional(''),
            "status" :          Optional(None),     # "under_review", "approved" or
↪"rejected"
            "close_pr" :        Optional(False)"
          }
```

OUTPUT:

```
id : <id_given_in_input>
result : True
error : null
```

3.4.5 API access for web views

Kallithea HTTP entry points can also be accessed without login using bearer authentication by including this header with the request:

```
Authentication: Bearer <api_key>
```

Alternatively, the API key can be passed in the URL query string using `?api_key=<api_key>`, though this is not recommended due to the increased risk of API key leaks, and support will likely be removed in the future.

Exposing raw diffs is a good way to integrate with third-party services like code review, or build farms that can download archives.

CHAPTER 4

Developer guide

4.1 Contributing to Kallithea

Kallithea is developed and maintained by its users. Please join us and scratch your own itch.

4.1.1 Infrastructure

The main repository is hosted on Our Own Kallithea (aka OOK) at https://kallithea-scm.org/repos/kallithea/, our self-hosted instance of Kallithea.

Please use the mailing list to send patches or report issues.

We use Weblate to translate the user interface messages into languages other than English. Join our project on Hosted Weblate to help us. To register, you can use your Bitbucket or GitHub account. See *Translations* for more details.

4.1.2 Getting started

To get started with Kallithea development run the following commands in your bash shell:

```
hg clone https://kallithea-scm.org/repos/kallithea
cd kallithea
python3 -m venv venv
. venv/bin/activate
pip install --upgrade pip setuptools
pip install --upgrade -e . -r dev_requirements.txt python-ldap python-pam
kallithea-cli config-create my.ini
kallithea-cli db-create -c my.ini --user=user --email=user@example.com --
↪password=password --repos=/tmp
kallithea-cli front-end-build
gearbox serve -c my.ini --reload &
firefox http://127.0.0.1:5000/
```

4.1.3 Contribution flow

Starting from an existing Kallithea clone, make sure it is up to date with the latest upstream changes:

```
hg pull
hg update
```

Review the *Contribution guidelines* and *Coding guidelines*.

If you are new to Mercurial, refer to Mercurial Quick Start and Beginners Guide on the Mercurial wiki.

Now, make some changes and test them (see *Internal dependencies*). Don't forget to add new tests to cover new functionality or bug fixes.

For documentation changes, run `make html` from the `docs` directory to generate the HTML result, then review them in your browser.

Before submitting any changes, run the cleanup script:

```
./scripts/run-all-cleanup
```

When you are completely ready, you can send your changes to the community for review and inclusion, via the mailing list (via `hg email`).

4.1.4 Internal dependencies

We try to keep the code base clean and modular and avoid circular dependencies. Code should only invoke code in layers below itself.

Imports should import whole modules `from` their parent module, perhaps `as` a shortened name. Avoid imports `from` modules.

To avoid cycles and partially initialized modules, `__init__.py` should *not* contain any non-trivial imports. The top level of a module should *not* be a facade for the module functionality.

Common code for a module is often in `base.py`.

The important part of the dependency graph is approximately linear. In the following list, modules may only depend on modules below them:

`tests` Just get the job done - anything goes.

`bin/` & `config/` & `alembic/` The main entry points, defined in `setup.py`. Note: The TurboGears template use `config` for the high WSGI application - this is not for low level configuration.

`controllers/` The top level web application, with TurboGears using the `root` controller as entry point, and `routing` dispatching to other controllers.

`templates/**.html` The "view", rendering to HTML. Invoked by controllers which can pass them anything from lower layers - especially `helpers` available as `h` will cut through all layers, and `c` gives access to global variables.

`lib/helpers.py` High level helpers, exposing everything to templates as `h`. It depends on everything and has a huge dependency chain, so it should not be used for anything else. TODO.

`controllers/base.py` The base class of controllers, with lots of model knowledge.

`lib/auth.py` All things related to authentication. TODO.

`lib/utils.py` High level utils with lots of model knowledge. TODO.

lib/hooks.py Hooks into "everything" to give centralized logging to database, cache invalidation, and extension handling. TODO.

model/ Convenience business logic wrappers around database models.

model/db.py Defines the database schema and provides some additional logic.

model/scm.py All things related to anything. TODO.

SQLAlchemy Database session and transaction in thread-local variables.

lib/utils2.py Low level utils specific to Kallithea.

lib/webutils.py Low level generic utils with awareness of the TurboGears environment.

TurboGears Request, response and state like i18n gettext in thread-local variables. External dependency with global state - usage should be minimized.

lib/vcs/ Previously an independent library. No awareness of web, database, or state.

lib/* Various "pure" functionality not depending on anything else.

__init__ Very basic Kallithea constants - some of them are set very early based on `.ini`.

This is not exactly how it is right now, but we aim for something like that. Especially the areas marked as TODO have some problems that need untangling.

4.1.5 Running tests

After finishing your changes make sure all tests pass cleanly. Run the testsuite by invoking `py.test` from the project root:

```
py.test
```

Note that on unix systems, the temporary directory (`/tmp` or where `$TMPDIR` points) must allow executable files; Git hooks must be executable, and the test suite creates repositories in the temporary directory. Linux systems with /tmp mounted noexec will thus fail.

Tests can be run on PostgreSQL like:

```
sudo -u postgres createuser 'kallithea-test' --pwprompt  # password password
sudo -u postgres createdb 'kallithea-test' --owner 'kallithea-test'
REUSE_TEST_DB='postgresql://kallithea-test:password@localhost/kallithea-test' py.test
```

Tests can be run on MariaDB/MySQL like:

```
echo "GRANT ALL PRIVILEGES ON \`kallithea-test\`.* TO 'kallithea-test'@'localhost'␣
↪IDENTIFIED BY 'password'" | sudo -u mysql mysql
TEST_DB='mysql://kallithea-test:password@localhost/kallithea-test?charset=utf8mb4' py.
↪test
```

You can also use `tox` to run the tests with all supported Python versions.

When running tests, Kallithea generates a *test.ini* based on template values in *kallithea/tests/conftest.py* and populates the SQLite database specified there.

It is possible to avoid recreating the full test database on each invocation of the tests, thus eliminating the initial delay. To achieve this, run the tests as:

```
gearbox serve -c /tmp/kallithea-test-XXX/test.ini --pid-file=test.pid --daemon
KALLITHEA_WHOOSH_TEST_DISABLE=1 KALLITHEA_NO_TMP_PATH=1 py.test
kill -9 $(cat test.pid)
```

In these commands, the following variables are used:

```
KALLITHEA_WHOOSH_TEST_DISABLE=1 - skip whoosh index building and tests
KALLITHEA_NO_TMP_PATH=1 - disable new temp path for tests, used mostly for testing_
↪vcs_operations
```

You can run individual tests by specifying their path as argument to py.test. py.test also has many more options, see *py.test -h*. Some useful options are:

```
-k EXPRESSION         only run tests which match the given substring
                      expression. An expression is a python evaluable
                      expression where all names are substring-matched
                      against test names and their parent classes. Example:
-x, --exitfirst       exit instantly on first error or failed test.
--lf                  rerun only the tests that failed at the last run (or
                      all if none failed)
--ff                  run all tests but run the last failures first. This
                      may re-order tests and thus lead to repeated fixture
                      setup/teardown
--pdb                 start the interactive Python debugger on errors.
-s, --capture=no      don't capture stdout (any stdout output will be
                      printed immediately)
```

Performance tests

A number of performance tests are present in the test suite, but they are not run in a standard test run. These tests are useful to evaluate the impact of certain code changes with respect to performance.

To run these tests:

```
env TEST_PERFORMANCE=1 py.test kallithea/tests/performance
```

To analyze performance, you could install pytest-profiling, which enables the –profile and –profile-svg options to py.test.

4.1.6 Contribution guidelines

Kallithea is GPLv3 and we assume all contributions are made by the committer/contributor and under GPLv3 unless explicitly stated. We do care a lot about preservation of copyright and license information for existing code that is brought into the project.

Contributions will be accepted in most formats – such as commits hosted on your own Kallithea instance, or patches sent by email to the kallithea-general mailing list.

Make sure to test your changes both manually and with the automatic tests before posting.

We care about quality and review and keeping a clean repository history. We might give feedback that requests polishing contributions until they are “perfect”. We might also rebase and collapse and make minor adjustments to your changes when we apply them.

We try to make sure we have consensus on the direction the project is taking. Everything non-sensitive should be discussed in public – preferably on the mailing list. We aim at having all non-trivial changes reviewed by at least one other core developer before pushing. Obvious non-controversial changes will be handled more casually.

There is a main development branch ("default") which is generally stable so that it can be (and is) used in production. There is also a "stable" branch that is almost exclusively reserved for bug fixes or trivial changes. Experimental changes should live elsewhere (for example in a pull request) until they are ready.

4.1.7 Coding guidelines

We don't have a formal coding/formatting standard. We are currently using a mix of Mercurial's (https://www.mercurial-scm.org/wiki/CodingStyle), pep8, and consistency with existing code. Run `scripts/run-all-cleanup` before committing to ensure some basic code formatting consistency.

We support Python 3.6 and later.

We try to support the most common modern web browsers. IE9 is still supported to the extent it is feasible, IE8 is not.

We primarily support Linux and OS X on the server side but Windows should also work.

HTML templates should use 2 spaces for indentation ... but be pragmatic. We should use templates cleverly and avoid duplication. We should use reasonable semantic markup with element classes and IDs that can be used for styling and testing. We should only use inline styles in places where it really is semantic (such as `display: none`).

JavaScript must use `;` between/after statements. Indentation 4 spaces. Inline multiline functions should be indented two levels – one for the `()` and one for `{}`. Variables holding jQuery objects should be named with a leading `$`.

Commit messages should have a leading short line summarizing the changes. For bug fixes, put `(Issue #123)` at the end of this line.

Use American English grammar and spelling overall. Use English title case for page titles, button labels, headers, and 'labels' for fields in forms.

Template helpers (that is, everything in `kallithea.lib.helpers`) should only be referenced from templates. If you need to call a helper from the Python code, consider moving the function somewhere else (e.g. to the model).

Notes on the SQLAlchemy session

Each HTTP request runs inside an independent SQLAlchemy session (as well as in an independent database transaction). `Session` is the session manager and factory. `Session()` will create a new session on-demand or return the current session for the active thread. Many database operations are methods on such session instances. The session will generally be removed by TurboGears automatically.

Database model objects (almost) always belong to a particular SQLAlchemy session, which means that SQLAlchemy will ensure that they're kept in sync with the database (but also means that they cannot be shared across requests).

Objects can be added to the session using `Session().add`, but this is rarely needed:

- When creating a database object by calling the constructor directly, it must explicitly be added to the session.
- When creating an object using a factory function (like `create_repo`), the returned object has already (by convention) been added to the session, and should not be added again.
- When getting an object from the session (via `Session().query` or any of the utility functions that look up objects in the database), it's already part of the session, and should not be added again. SQLAlchemy monitors attribute modifications automatically for all objects it knows about and syncs them to the database.

SQLAlchemy also flushes changes to the database automatically; manually calling `Session().flush` is usually only necessary when the Python code needs the database to assign an "auto-increment" primary key ID to a freshly created model object (before flushing, the ID attribute will be `None`).

Debugging

A good way to trace what Kallithea is doing is to keep an eye on the output on stdout/stderr of the server process. Perhaps change `my.ini` to log at `DEBUG` or `INFO` level, especially `[logger_kallithea]`, but perhaps also other loggers. It is often easier to add additional `log` or `print` statements than to use a Python debugger.

Sometimes it is simpler to disable `errorpage.enabled` and perhaps also `trace_errors.enable` to expose raw errors instead of adding extra processing. Enabling `debug` can be helpful for showing and exploring tracebacks in the browser, but is also insecure and will add extra processing.

TurboGears2 DebugBar

It is possible to enable the TurboGears2-provided DebugBar, a toolbar overlayed over the Kallithea web interface, allowing you to see:

- timing information of the current request, including profiling information
- request data, including GET data, POST data, cookies, headers and environment variables
- a list of executed database queries, including timing and result values

DebugBar is only activated when `debug = true` is set in the configuration file. This is important, because the DebugBar toolbar will be visible for all users, and allow them to see information they should not be allowed to see. Like is anyway the case for `debug = true`, do not use this in production!

To enable DebugBar, install `tgext.debugbar` and `kajiki` (typically via `pip`) and restart Kallithea (in debug mode).

4.1.8 Thank you for your contribution!

4.2 Translations

Translations are available on Hosted Weblate at the following URL:

https://hosted.weblate.org/projects/kallithea/kallithea/

Registered users may contribute to the existing languages, or request a new language translation.

4.2.1 Translating using Weblate

Weblate offers a simple and easy to use interface featuring glossary, machine translation, suggestions based on similar translations in other projects, automatic checks etc. Weblate imports the source code tree directly from the version control system, and commits edits back from time to time.

When registering at Weblate, make sure you use the name and email address you prefer to be used when your changes are committed. We can and probably will amend changesets coming from Weblate, but having things right from the beginning makes things easier.

Weblate performs sanity checks all the time and tries to prevent you from ignoring them. Most common mistakes are inconsistent punctuation, whitespace, missing or extra format parameters, untranslated strings copied into the translation. Please perform necessary corrections when they're needed, or override the false positives.

4.2.2 Merging translations from Weblate (admin-only)

Weblate rebases its changes every time it pulls from our repository. Pulls are triggered by a web hook from Our Own Kallithea every time it receives new commits. Usually merging the new translations is a straightforward process consisting of a pull from the Weblate-hosted repository which is available under the Data Exports tab in the Weblate interface.

Weblate tries to minimise the number of commits, but that doesn't always work, especially when two translators work with different languages at more or less the same time. It makes sense sometimes to re-order or fold commits by the same author when they touch just the same language translation. That, however, may confuse Weblate sometimes, in which case it should be manually convinced it has to discard the commits it created by using its administrative interface.

4.2.3 Regenerating translations after source code changes (admin-only)

When the Kallithea source code changes, both the location as the content of translation strings can change. It is therefore necessary to regularly regenerate the *kallithea.pot* file containing these strings, as well as aligning the translation files (**.po*).

First update the translation strings:

```
python3 setup.py extract_messages
```

Then regenerate the translation files. This could either be done with *python3 setup.py update_catalog* or with *msgmerge* from the *gettext* package. As Weblate is also touching these translation files, it is preferred to use the same tools (*msgmerge*) and settings as Weblate to minimize the diff:

```
find kallithea/i18n -name kallithea.po | xargs -I '{}' \
    msgmerge --width=76 --backup=none --previous --update '{}' \
    kallithea/i18n/kallithea.pot
```

4.2.4 Manual creation of a new language translation

In the prepared development environment, run the following to ensure all translation strings are extracted and up-to-date:

```
python3 setup.py extract_messages
```

Create new language by executing following command:

```
python3 setup.py init_catalog -l <new_language_code>
```

This creates a new translation under directory *kallithea/i18n/<new_language_code>* based on the translation template file, *kallithea/i18n/kallithea.pot*.

Edit the new PO file located in *LC_MESSAGES* directory with poedit or your favorite PO files editor. After you finished with the translations, check the translation file for errors by executing:

```
msgfmt -f -c kallithea/i18n/<new_language_code>/LC_MESSAGES/<updated_file.po>
```

Finally, compile the translations:

```
python3 setup.py compile_catalog -l <new_language_code>
```

4.2.5 Manually updating translations

Extract the latest versions of strings for translation by running:

```
python3 setup.py extract_messages
```

Update the PO file by doing:

```
python3 setup.py update_catalog -l <new_language_code>
```

Edit the newly updated translation file. Repeat all steps after the *init_catalog* step from the 'new translation' instructions above.

4.2.6 Testing translations

Edit *kallithea/tests/conftest.py* and set *i18n.lang* to *<new_language_code>* and run Kallithea tests by executing:

```
py.test
```

4.2.7 Managing translations with scripts/i18n tooling

The general idea with the `scripts/i18n` tooling is to keep changes in the main repository focussed on actual and reviewable changes with minimal noise. Noisy generated or redundant localization changes (that are useful when translations) are contained in the `kallithea-i18n` repo on the `i18n` branch. The translation files in the main repository have no line numbers, no untranslated entries, no fuzzy entries, no unused entries, and no constantly changing records of "latest" this and that (name, date, version, etc).

The branches in the main repo (`default` and `stable`) will thus only have stripped `.pot` and `.po` files: an (almost) empty `kallithea/i18n/kallithea.pot` file, and minimal `.po` files. There are no binary `.mo` files in any repo - these are only generated when packaging for release (or locally if installing from source).

Generally, `kallithea/i18n/` should not be changed on the `default` and `stable` branches at all. The `i18n` branch should *only* change `kallithea/i18n/` . If there are changesets with exceptions from that, these changesets should probably be grafted/redone in the "right" place.

The basic flow is thus:

0. All weblate translation is done on the `i18n` branch which generally is based on the `stable` branch.
1. Graft the essential part of all new changes on the `i18n` branch to `stable` (while normalizing to current stripped state of stable).
2. Merge from `stable` to `i18n` (while normalizing to the resulting unstripped and fully `msgmerge`'d state and `.pot`-updating state).
3. Verify that the content of the `i18n` branch will give exactly the content of the `stable` branch after stripping. If there is a diff, something has to be fixed in one way or the other ... and the whole process should probably be redone.

Translate

First land full translation changes in the `kallithea-i18n` repo on the `i18n` branch. That can be done in pretty much any way you want. If changes for some reason have to be grafted or merged, there might be odd conflicts due to all the noise. Conflicts on the full `i18n` branch can perhaps be resolved more easily using non-stripping normalization before merging:

```
python3 setup.py extract_messages && cp kallithea/i18n/kallithea.pot full.pot && hg␣
↪revert kallithea/i18n/kallithea.pot -r .
hg resolve kallithea/i18n/ --tool X --config merge-tools.X.executable=python3 --
↪config merge-tools.X.args='scripts/i18n normalized-merge --merge-pot-file full.pot
↪$local $base $other $output'
```

Land in main repository - stripped

When the full i18n changes have landed on the `i18n` branch, prepare to land them on `stable`:

```
hg up -cr stable
python3 setup.py extract_messages && cp kallithea/i18n/kallithea.pot full.pot && hg␣
↪revert kallithea/i18n/kallithea.pot
```

Consider all new `i18n` changes since last merge from `stable`:

```
hg log -G --style compact -r 'only("i18n", children(::stable))'
```

Graft them one by one (or in collapsed chunks) while normalizing.

If the graft has conflicts, use the `scripts/i18n` normalization tool to apply `msgmerge` and strip before doing 3-way merge and resolving conflicts:

```
hg resolve kallithea/i18n/ --tool X --config merge-tools.X.executable=python3 --
↪config merge-tools.X.args='scripts/i18n normalized-merge --merge-pot-file full.pot -
↪-strip $local $base $other $output'
```

When all conflicts have been resolved, continue the graft:

```
hg graft --continue
```

Then make sure any non-conflicting files are normalized and stripped too:

```
scripts/i18n normalize-po-files --strip --merge-pot-file full.pot kallithea/i18n/*/LC_
↪MESSAGES/kallithea.po
hg ci --amend --config ui.editor=true
```

When things have been grafted to the `stable` branch, clean up history if necessary: clean up the author and commit message when necessary, and perhaps merge multiple changesets from same contributor.

Merge back to `i18n`

For any i18n changes that for some reason have been done on the `stable` branch, apply them manually on the `i18n` branch too - perhaps by grafting and editing manually. The merge done in this step will *not* take care of it. If the verification step done a bit later points out that something has been missed, strip and go back to this point.

Then merge back to the `i18n` branch using normalization while keeping the full `.po` files, and updating the full `.pot` and `.po` to current state:

```
hg up -cr i18n
hg merge stable --tool internal:fail
hg revert kallithea/i18n/*/LC_MESSAGES/*.po -r .
hg resolve -m kallithea/i18n/*/LC_MESSAGES/*.po
hg resolve -l  # verify all conflicts have been resolved
python3 setup.py extract_messages && cp kallithea/i18n/kallithea.pot full.pot
```

(continues on next page)

(continued from previous page)

```
scripts/i18n normalize-po-files --merge-pot-file full.pot kallithea/i18n/*/LC_
↪MESSAGES/kallithea.po
hg commit  # "Merge from stable"
```

Note: `normalize-po-files` can also pretty much be done manually with:

```
for po in kallithea/i18n/*/LC_MESSAGES/kallithea.po; do msgmerge --width=76 --
↪backup=none --previous --update $po full.pot ; done
```

Note: Additional merges from `stable` to `i18n` can be done any time.

Verify

Verify things are in sync between the full `i18n` branch and the stripped `stable` branch:

```
hg up -cr stable
hg revert -a -r i18n
python3 setup.py extract_messages && cp kallithea/i18n/kallithea.pot full.pot && hg␣
↪revert kallithea/i18n/kallithea.pot
scripts/i18n normalize-po-files --strip --merge-pot-file full.pot kallithea/i18n/*/LC_
↪MESSAGES/kallithea.po
hg diff
```

If there is a diff, figure out where it came from, go back and fix the root cause, and redo the graft/merge.

Push

The changes on the `stable` branch should now be ready for pushing - verify the actual changes with a thorough review of:

```
hg out -pvr stable
```

When `stable` changes have been pushed, also push the `i18n` branch to the `kallithea-i18n` repo so Weblate can see it.

4.3 Database schema changes

Kallithea uses Alembic for *database migrations* (upgrades and downgrades).

If you are developing a Kallithea feature that requires database schema changes, you should make a matching Alembic database migration script:

1. *Create a Kallithea configuration and database* for testing the migration script, or use existing `development.ini` setup.

   Ensure that this database is up to date with the latest database schema *before* the changes you're currently developing. (Do not create the database while your new schema changes are applied.)

2. Create a separate throwaway configuration for iterating on the actual database changes:

   ```
   kallithea-cli config-create temp.ini
   ```

   Edit the file to change database settings. SQLite is typically fine, but make sure to change the path to e.g. `temp.db`, to avoid clobbering any existing database file.

3. Make your code changes (including database schema changes in `db.py`).
4. After every database schema change, recreate the throwaway database to test the changes:

```
rm temp.db
kallithea-cli db-create -c temp.ini --repos=/var/repos --user=doe --email␣
↪doe@example.com --password=123456 --no-public-access --force-yes
kallithea-cli repo-scan -c temp.ini
```

5. Once satisfied with the schema changes, auto-generate a draft Alembic script using the development database that has *not* been upgraded. (The generated script will upgrade the database to match the code.)

```
alembic -c development.ini revision -m "area: add cool feature" --autogenerate
```

6. Edit the script to clean it up and fix any problems.

   Note that for changes that simply add columns, it may be appropriate to not remove them in the downgrade script (and instead do nothing), to avoid the loss of data. Unknown columns will simply be ignored by Kallithea versions predating your changes.

7. Run `alembic -c development.ini upgrade head` to apply changes to the (non-throwaway) database, and test the upgrade script. Also test downgrades.

   The included `development.ini` has full SQL logging enabled. If you're using another configuration file, you may want to enable it by setting `level = DEBUG` in section `[handler_console_sql]`.

The Alembic migration script should be committed in the same revision as the database schema (`db.py`) changes.

See the Alembic documentation for more information, in particular the tutorial and the section about auto-generating migration scripts.

4.3.1 Troubleshooting

- If `alembic --autogenerate` responds "Target database is not up to date", you need to either first use Alembic to upgrade the database to the most recent version (before your changes), or recreate the database from scratch (without your schema changes applied).