ATLASSIAN DOCUMENTATION

bitbucket Documentation Home

bitbucket is a hosting site for the distributed version control systems (DVCS) Git and Mercurial. The service offering includes an issue tracker and wiki, as well as integration with a number of popular services such as Basecamp, Flowdock and Twitter.

Where to begin

If you are entirely new to using bitbucket you should work through the bitbucket 101 tutorial.

Popular Topics

- Managing your account
- bitbucket permissions
- Importing code
- Administering your repo
- Using your repo
- Using the wiki
- Tracking your issues

Developer Resources

You can customize and extend bitbucket by:

Working with REST – Use bitbucket's REST APIs to interact programmatically with a repository, wiki, issue tracker and with other users.

Develop a custom broker – Brokers are Python scripts that allow repositories to integrate with external services such as BaseCamp and Twitter.

Full Table of Contents

▼ Click to list the entire contents of this space.

- bitbucket 101
  - Set up Git and Mercurial
  - Create an Account and a Git Repo!
  - Clone Your Git Repo and Add Source Files
  - Fork a Repo, Compare Code, and Create a Pull Request
  - Add Users, Set Permissions, and Review Account Plans
  - Set up a Wiki and an Issue Tracker
  - Set up SSH for Git
  - Set up SSH for Mercurial
  - Mac Users: SourceTree a Free Git and Mercurial GUI
- Importing code from an existing project
- Using the SSH protocol with bitbucket
  - Configuring Multiple SSH Identities for GitBash, Mac OSX, & Linux
  - Configuring Multiple SSH Identities for TortoiseHg
  - Troubleshooting SSH Issues
- Repository privacy, permissions, and more
- Managing your account
  - Managing Repository Users
  - Managing the user groups for your Repositories
  - Setting Your Email Preferences
  - Deleting Your Account
  - Renaming Your Account
  - Using your Own bitbucket Domain Name
- Administrating a repository
  - Granting Groups Access to a Repository
  - Granting Users Access to a Repository
  - Managing bitbucket Services
    - Setting Up the Bitbucket Basecamp Service
    - Setting Up the Bitbucket CIA.vc Service
    - Setting Up the Bitbucket Email Diff Service
    - Setting Up the Bitbucket Email Service
    - Setting Up the Bitbucket Flowdock Service
    - Setting Up the Bitbucket FogBugz Service
    - Setting Up the Bitbucket FriendFeed Service
    - Setting Up the Bitbucket Geocommit Service

bitbucket 101

If you are new to hosting your code, DVCS code management, or either Git or Mercurial, this bitbucket 101 tutorial gives you a taste all of them. In this tutorial, you'll first install both Git and Mercurial for your operating system. You'll do some work using both Git and then Mercurial. Throughout, you'll use the hosted code management system that is bitbucket. The tutorial consists of the following pages:

- Set up Git and Mercurial
- Create an Account and a Git Repo!
- Clone Your Git Repo and Add Source Files
- Fork a Repo, Compare Code, and Create a Pull Request
- Add Users, Set Permissions, and Review Account Plans
- Set up a Wiki and an Issue Tracker
- Set up SSH for Git
- Set up SSH for Mercurial
- Mac Users: SourceTree a Free Git and Mercurial GUI

The tutorial teaches you some simple DVCS workflows using basic Git and Mercurial commands. You don't really need to know anything about Git or Mercurial to work through the tutorial. If you are a beginner, you may not be comfortable working this way. If you prefer to learn the tools before working through this tutorial, you can learn more about Git here. To learn more about Mercurial, you should start here.

How to work through the tutorial

If you are totally new to DVCS and/or bitbucket, you should work through each page sequentially as each new page builds on the material from the previous pages At the end of each page is a **Next** heading that tells you where to go next. If you get lost, you can use the navigation bar (to your left) to locate the next page. If you feel confident skipping pages or just going to pages you need, feel free to do that too.

If you are a total beginner, you should allow at least a couple of hours to work through the entire tutorial. If you are experienced or just skimming pages, much of this will be familiar to you and it should not take too long.

 Start the tutorial with Set up Git and Mercurial.

Skip the getting started stuff

You can skip the getting started bit if you want. Other things you might be interested in:

- frequently asked questions and answers
- the bitbucket Google group
- how to use your repository
- the bitbucket blog

Set up Git and Mercurial

To use bitbucket, you need to install a DVCS tool on the computer where you write your code. Typically, this computer is a machine physically close to you like your home or work computer. This is your local machine or system. You also might write or deploy code to a remote machine – for example a lab computer or a server in a data center. You may also need a DVCS tool on that machine too. This tutorial refers to the typical case, your local system, but the instructions for both cases.

bitbucket supports two DVCS tools, Git or Mercurial. These tools run on all modern operating systems. You can install either or both Git and Mercurial on any of their supported operating systems. For Git, bitbucket supports 1.6.6 or later and Mercurial version 1.7 or later. Mercurial also requires (depends on) the Python programming language. The installation process takes care of making sure you get the correct version of Python.

Since you can use both Git and Mercurial on the same machine, this page shows you how to install both because you need both to complete this tutorial. If you already have one or both of these tools installed, skip the instructions and go the next step in the tutorial.

> This documentation shows you how to install on Windows 7. If you want to install on Mac OSX or Linux, we have instructions for that too.

Step 1. Install msysgit

msysgit is the Microsoft Windows version of Git. Do the following to install Git on your Windows machine:

1. Download the msysgit installer package.
   You can either **Run** the installer directly from your browser's **File Download** dialog or you can **Save** the file to a folder on your computer. These instructions assume you run the installer when prompted by the downloader.
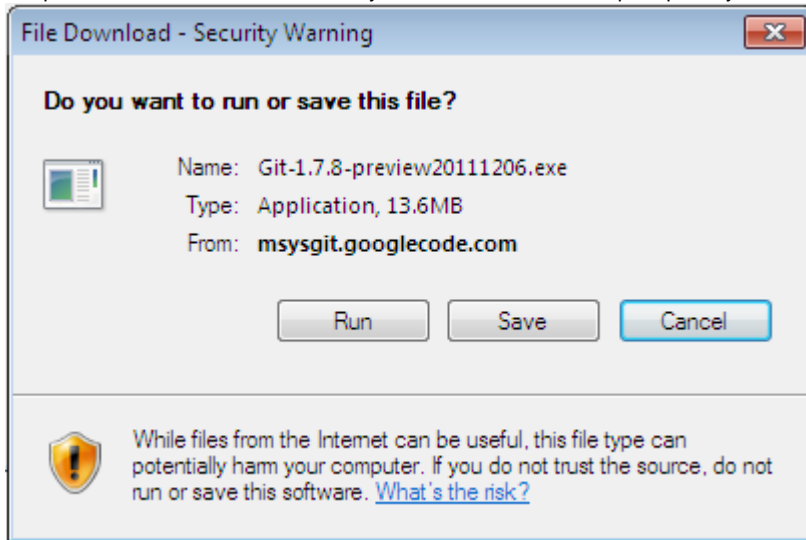
   If you downloaded the file to a folder on your local machine, you can also click the downloaded file's icon to run the installer. When you've successfully started the mysgit installer, you should see the **Git Setup** wizard screen:

   The version shown on your screen may be different than the one shown here of course. That is ok as long as you are installing a msysgit version 1.6.6 or later.

2. Press **Next** to move to the next page of the wizard.
   The setup displays the license agreement.

3. Press **Next** to accept the license agreement and continue.
   For this setup, use all the default setup values recommended by installer.

4. To accept all the defaults, press **Next** on each page of the dialog that comes after the license agreement.
5. Press **Finish** on the final page of the dialog to complete the installation.
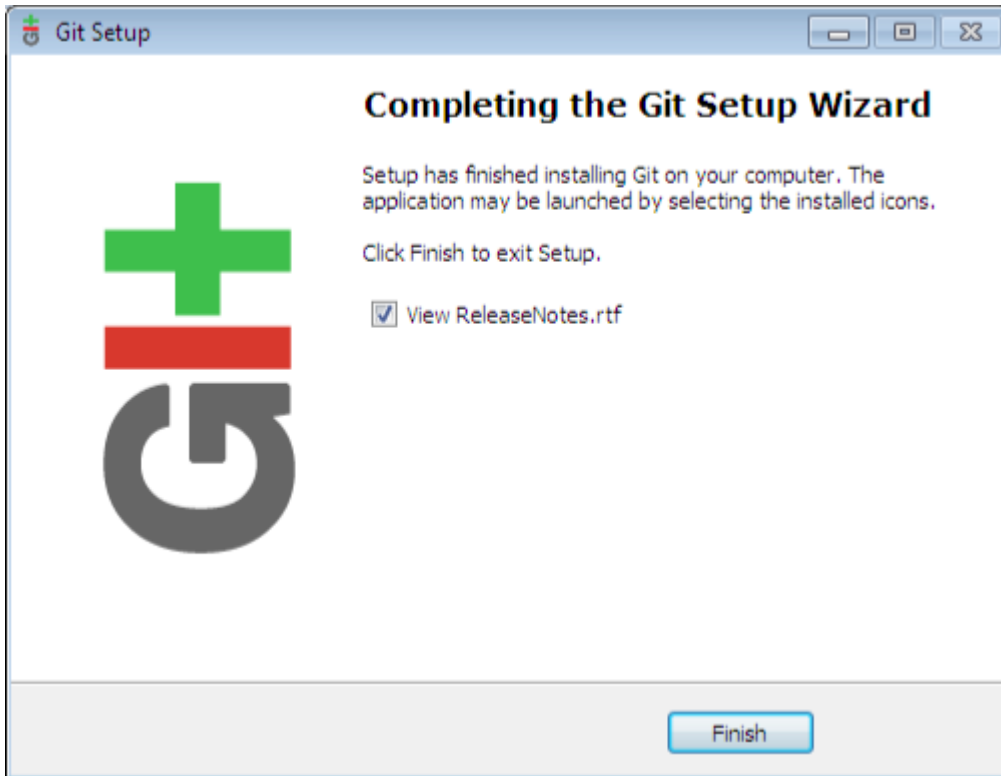
The system opens the release notes. You might want to skim them.

6. Close the release notes.
7. Open the **Git Bash** window by choosing **Start > All Programs > Git > Git Bash**.
   The system opens a command window:
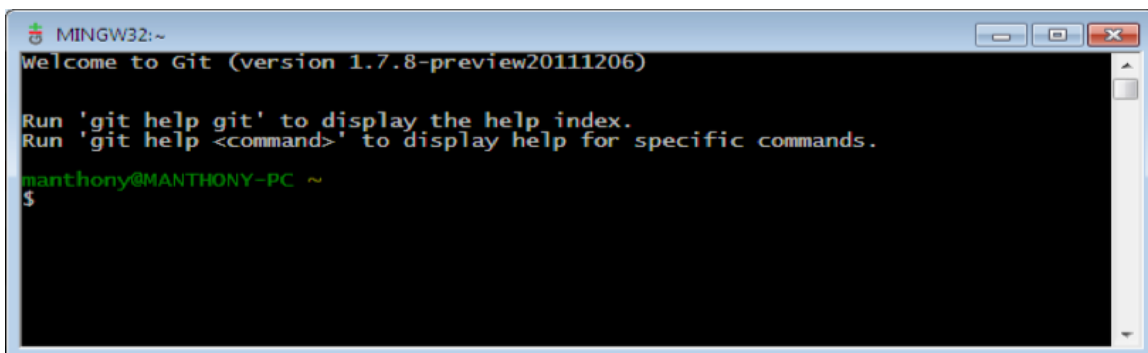
Using Git Bash, you can simply enter `the git` command lines that appears elsewhere in the bitbucket documentation. This documentation assumes you are familiar with a bash shell. If you want to use the Git GUI instead of Git Bash, you can, you'll need to learn that on your own though.

8. Configure your username using the following command:
   `git config --global user.name "FIRST_NAME LAST_NAME"`

9. Configure your email address using the following command:
   `git config --global user.email "MY_NAME@example.com"`

Step 3. Install Mercurial

TortoiseHg 2.2.1 is the Microsoft Windows version of Mercurial.

1. Download the all-in-one installer (MSI version) from the bitbucket repository (shameless promotion 😉) .

If you downloaded the file to a folder on your local machine, you can also click the downloaded file's icon to run the installer. When you've successfully started the mysgit installer, you should see the **TortoiseHg Setup** wizard screen:
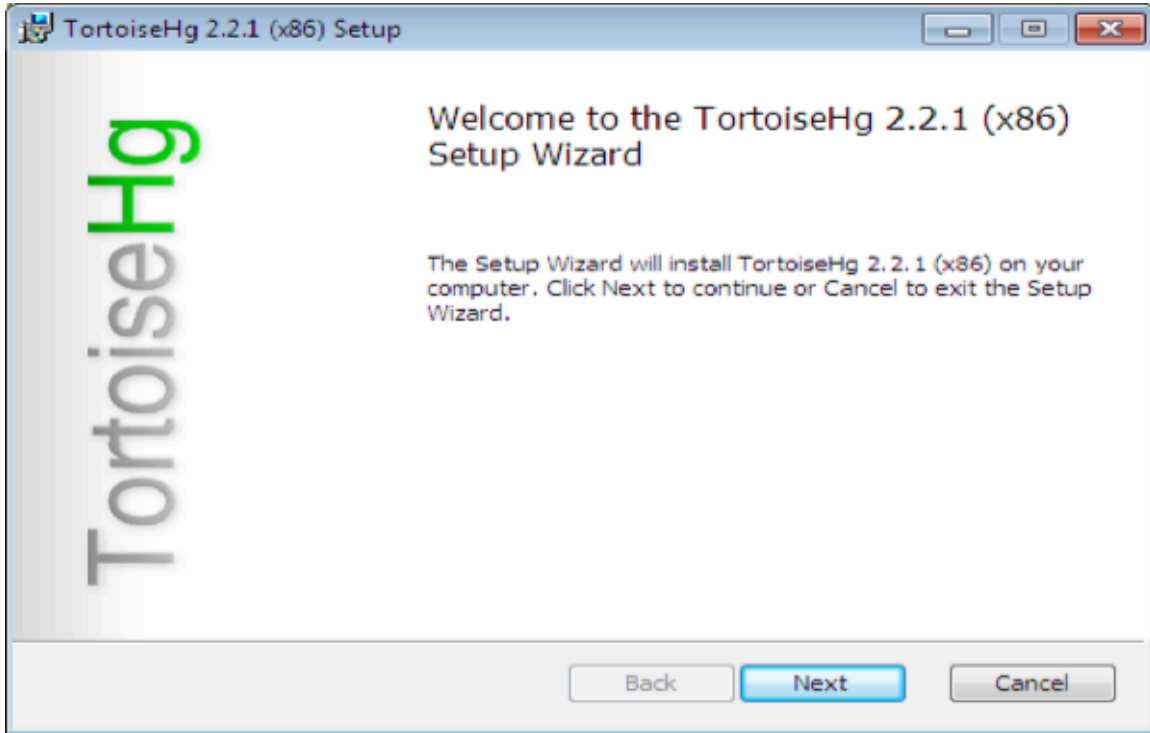
The version shown on your screen may be different than the one shown here of course. That is ok as long as you are installing a TortoiseHg that supports Mercurial version 1.7 or later.

2. Press **Next** to move to the next page of the wizard.
   The setup displays the license agreement.

3. Press **Next** to accept the license agreement and continue.
   For this setup, use all the default setup values recommended by installer.

4. To accept all the defaults, press **Next** on each page of the dialog that comes after the license agreement until the wizards prompts you to install.
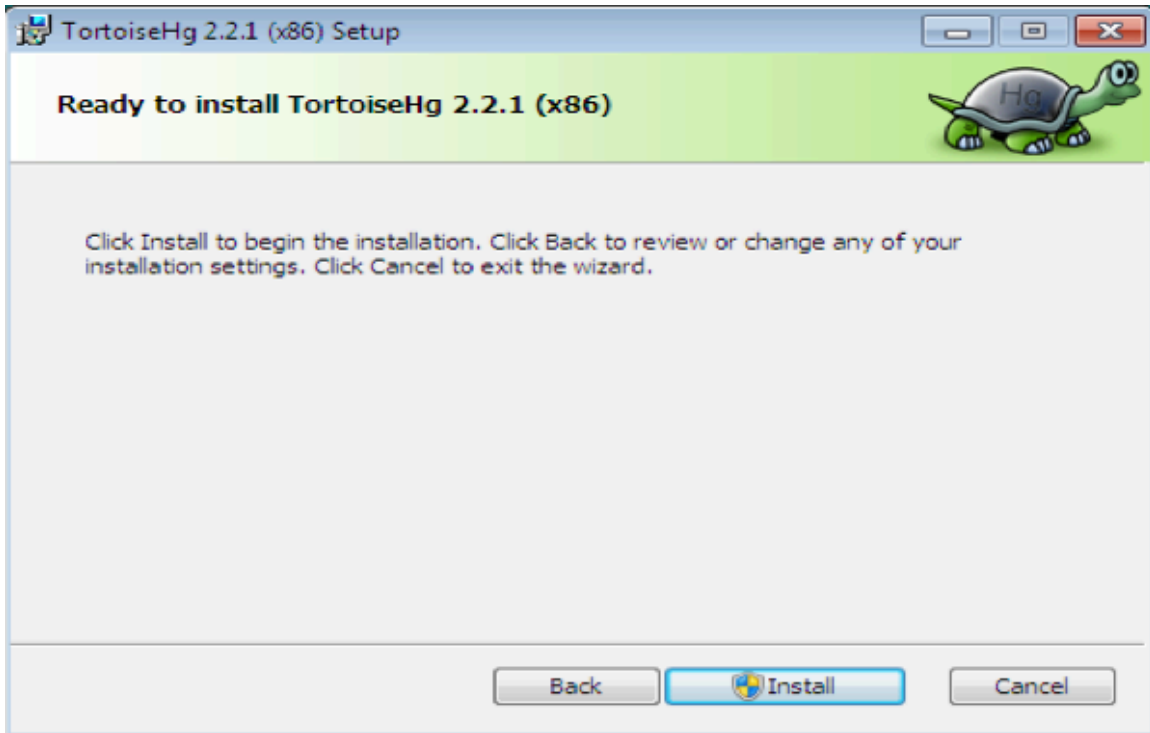
5. Press **Install** to install the software.
6. Press **Finish** on the final page of the dialog to complete the installation.
   You may need to restart your system for the installation to take effect.

Do the following to configure your global default username and email:

1. Right-click in your Desktop to open the context menu.
   The system displays the right hand context menu:
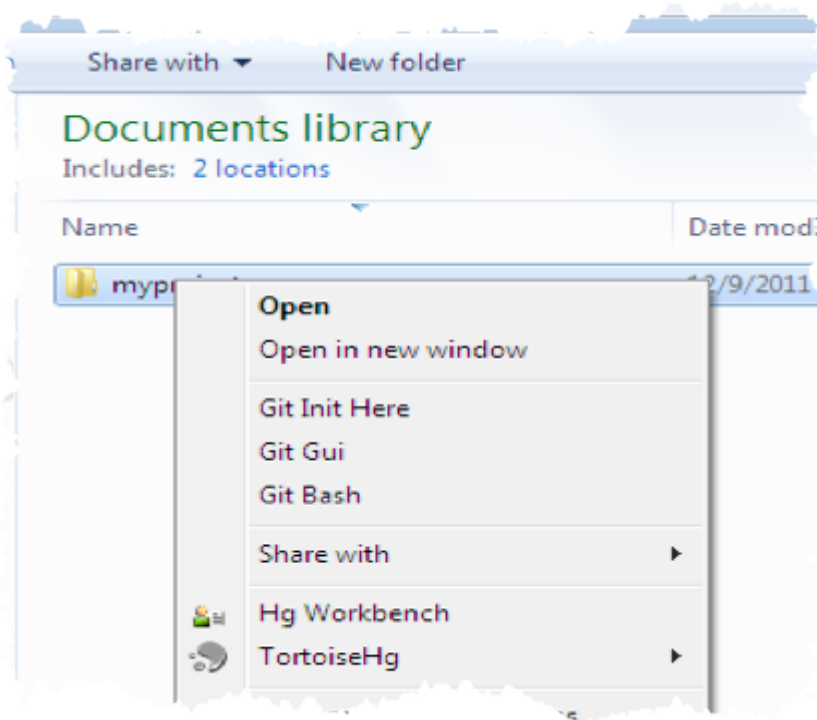
2. Click the **Hg Workbench** item.
   The system opens the **TortoiseHg Workbench** application.

3. Choose **File > Settings** to open the **TortoiseHg Settings** dialog.
4. Locate the **Commit** section on the left hand side and click it.
5. Fill in the **Username** value using the following format:
   firstname lastname<email>

   When done the dialog looks similar to this:
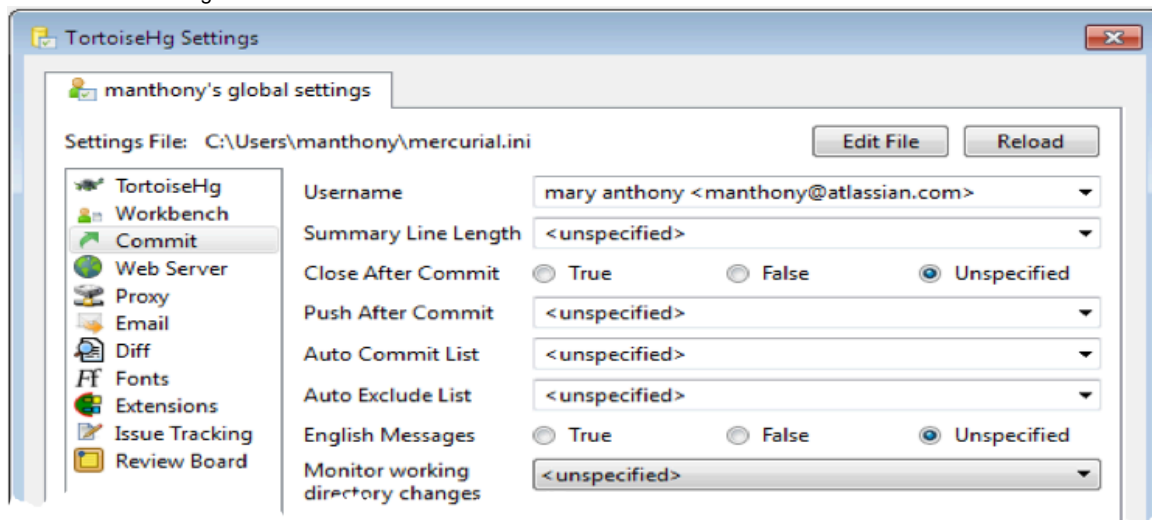
6. Press **OK** to save your changes.

Step 4. Install PuTTYgen

PuTTYgen is a free RSA and DSA key generation tool. (You'll learn more about RSA, DSA, and key generation later in this tutorial.) If you don't have PuTTYgen installed, do the following:

1. Download the proper version for your system.
   The PuTTYgen utility is a single executable file.
2. Move the `puttygen.exe` executable to the `C:\Program Files\TortoiseHg` folder.

Next Steps

The next step is to Create an Account and a Git Repo!

Create an Account and a Git Repo!

Now, you get to use bitbucket! On this page you'll create an account on bitbucket. Once you have an account, you can create a repository (or *repo* for short). Here are some quick facts about accounts, repos, and projects:

- both the username and email on an account must be unique
- each repo belongs to an account (sometimes called a username or user for short)
- the person who created the account is the repo owner
- the repo owner is the only person who can delete a repo
- a user (an account) can have an unlimited number of public and private repos
- a code project can consist of multiple repos across multiple accounts but can also be a single repo from a single account

Even if you only intend to work with other people's projects, knowing how to create a repo is useful. Finally, creating a repo is free so no reason not to do it.

Who needs an account?

Depending on what you want to do with bitbucket, you don't even need to create an account. You need an account if you are:

- a new bitbucket user working through the bitbucket 101
- a developer who is working on a project whose code happens to be in bitbucket
- a project owner/leader who wants to share a bitbucket project (repo) with others
- a developer who wants to contribute to a repo in bitbucket

Step 1. Create a bitbucket account

If you already have an account, skip this section and go to the next. You can create an account with the standard sign up or you can create one through OpenID (using an existing Google account or Yahoo for example). Regardless of which sign method you use, you must supply

Field	About what you are supplying
Username	A 30 character username. bitbucket appends this username to the URL for all the repos you create. For example, the username `atlassian_tutorial` has a corresponding bitbucket URL of `https://bitbucket.org/` `atlassian_tutorial`. You can use letters, numbers, and underscores in your username. Your username must be unique across the entire bitbucket site.
Email address	An email address that is unique across the entire bitbucket site. The system sends you a confirmation email.
Password	A combination of up to 128 characters. If you are using OpenId the system uses that password. You are responsible for ensuring that your account password is sufficiently complex to meet your personal security standards.

To use the a standard bitbucket sign up. Open your browser and do the following:

1. Open https://bitbucket.org/account/signup/ in your browser.
2. Complete the fields on the form.
3. Press **Sign up**.

If you want to do an OpenID sign up, do the following:

1. Open https://bitbucket.org/account/signup/ in your browser.
2. Click the **OpenID** link under the **Sign Up** button.
3. Choose a provider from the list.
4. Press **Log in**.
5. Follow the provider's prompts to complete the sign up process.

When you are done signing in, bitbucket places you in your account **Dashboard**. Take a second and look around at the UI. Across the top of each bitbucket page is a series of options that let you navigate around bitbucket. You can also navigate using these shortcuts:

Key or Combination	Action
c then **r**	Create repository.
i then **r**	Import repository.
g then **d**	Go to dashboard.
g then **a**	Go to account settings.
g then **i**	Go to inbox.
g then **e**	Explore bitbucket.

g then **r**	Go to **My repositories**.
/	Focus the site search. Puts your cursor in the search field.
esc	Dismisses the help dialog or removes the focus from a form field.
u	Goes back up the stack you just went down with the shortcuts. Like the back button in a browser, this takes you back through the bitbucket pages you just paged through.

Step 2. Create a Git repository (repo)

The next step is to create a repository (or repo) . Initially, the repo you create is going to be empty without any code in it. Do the following to create your repo:

1. Log into bitbucket (https://bitbucket.org/account/signin/).
   The system displays your account **Dashboard** which should be empty at this point:
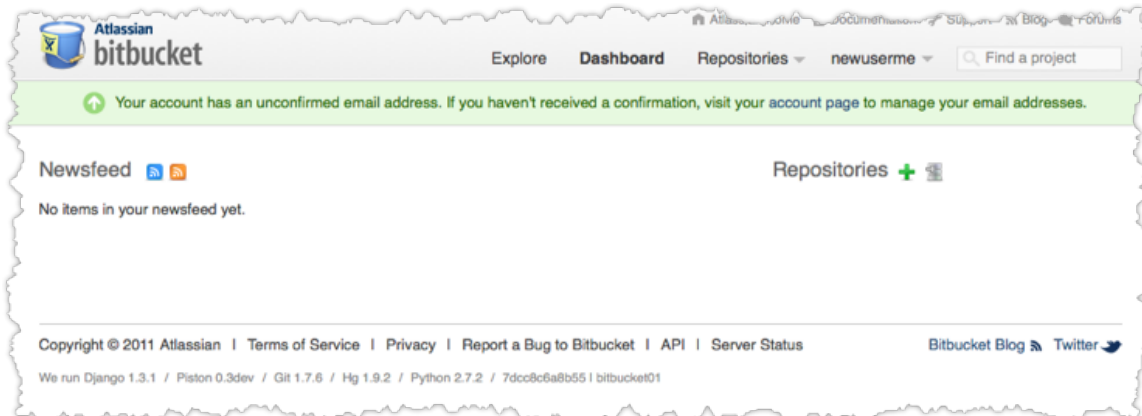
2. Click **Repositories > create repository** or click the green plus sign next to the **Repositories** heading on the page.
   The system displays the **Create new repository** page. Take some time to review the dialog's contents. With the exception of the **Repository type**, everything you enter on this page you can later change.
3. Enter `bb101repo` for the **Name** field.
   bitbucket uses this **Name** in the URL for your repo. For example, the username `atlassian_tutorial` has a repo name `jira-applinks`, the full URL for that repo is `https://bitbucket.org/atlassian_tutorial/jira-applinks`.
4. Leave the **Private** field checked.
   A private repo is only visible to you and those you give access (more about this later). If this box is unchecked, everyone can see your repo.
5. Under **Project management**, check both the **Issue tracking** and **Wiki** checkbox.
6. Check **Git** for the **Repository type**.
   You can't change this value later.
7. Set the **Language** field to **HTML/CSS**.
8. Enter a short **Description**.
9. Skip the **Website** field for now.
   The **Website** field lets you point users to an external website for your project.
10. Click **Create repository**.
    bitbucket creates your repo and displays the repo **Overview** page.

Step 3. Explore your new repo

Take some time to explorer the repo you have just created. You should be on the repo's **Overview** page:
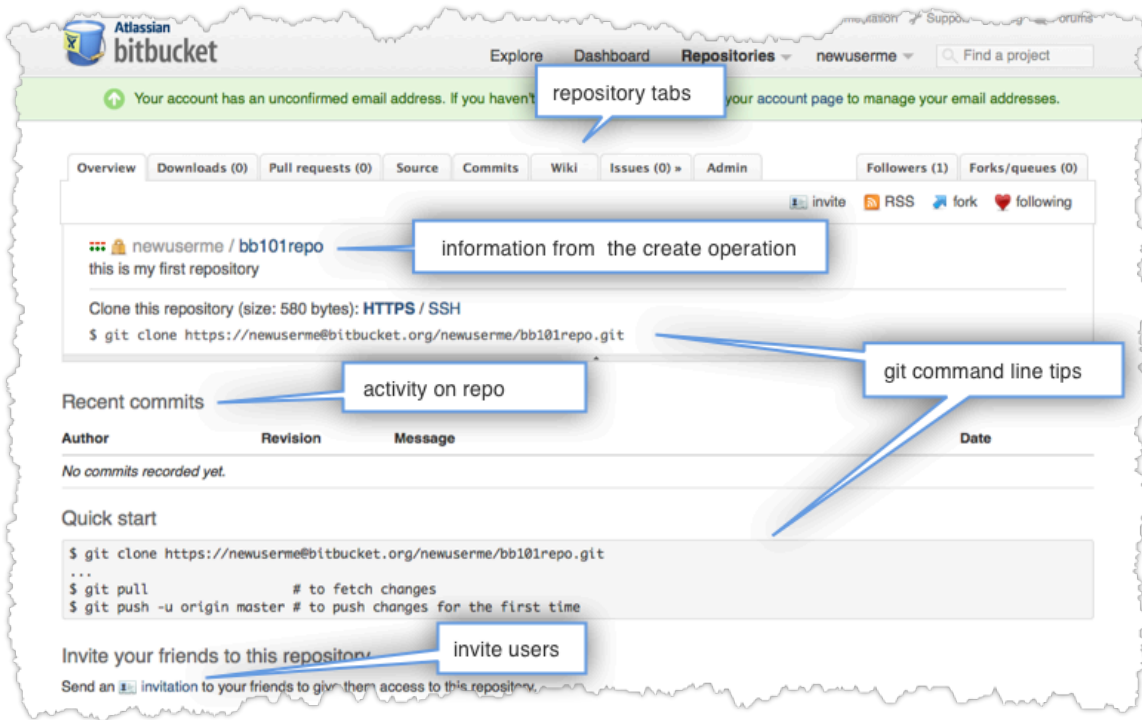
Across the top of each repo is a series of tabs that let you navigate around to your repos pages. Click the tabs to see what is behind each one. You can also navigate using the repo shortcuts.

Key or Combination	Action
r then **o**	Repository overview.
r then **d**	Repository downloads.
r then **q**	Repository patch series.
r then **p**	Repository pull requests.
r then **s**	Repository source.
r then **c**	Repository commits.
r then **w**	Repository wiki.
r then **i**	Repository issues.
r then **a**	Repository administration.

Return back to the **Overview** page. Now look at the **Recent commits** and you find there are **No commits recorded yet** (during your repo exploration you probably saw this message also). You have no commits because you have not created any content for your repo. Your repo is private and you have not invited anyone to the repo. So, the only person who can create or edit the repo's content right now is you, the repo *owner*.

Step 4. Confirm your email address

When you create an account, bitbucket sends an email to you to confirm your email. Confirming your email allows bitbucket to automatically match your user account to your future source code commits. So, do the following:

1. Open your email client (Gmail, Outlook, or whatever).
2. Find the email confirmation from bitbucket.
3. Open it.
4. Confirm the address.

Next

Now that you've created your first repo and explored it a bit, you are ready to add content to it.

Clone Your Git Repo and Add Source Files

On this page, you'll learn how to add code to your repo (the short form of "repository"). At this point, you have created a `bb101repo` which is a private repo that only you, the *owner*, can see or work in. This is not typical. Usually, you want some help on a project and you won't be the only one working in a repo. However, this is a tutorial and right now you need to learn without being distracted by other people. We'll add

more people to your project later.

> **ℹ Important reminders about these instructions**
>
> This tutorial teaches you a little Git first and later a little Mercurial. The git commands on this page are for a GitBash terminal. However, the commands are nearly identical in a Mac OSX terminal and in a Unbuntu Linux terminal. You shouldn't have any trouble following along but if you do, please comment on the page and we'll make corrections.
>
> All the operations that you do with Git have Mercurial equivalents and of course the reverse is also true. You are free to use Mercurial exclusively with this tutorial if you want, but you'll need to do that on your own.

Step 1. Clone your repo to your local system

Open a browser and a terminal window on your local system (the system you code on). Then, do the following:

1. Bring up the terminal window on your desktop and navigate to your home (~) directory.
   As you work with code, you will find that you have multiple repos that you work in. It is a good idea to create a directory to contain all those repos. Generally, this is your home directory but it can be anywhere you want it to be.
2. Create a directory to contain your repos.

```
    mkdir repos
```

3. If you haven't already done so, go to your browser, open up Bitbucket and log into your Bitbucket account.
4. Go to your `bb101repo` **Overview** page.
5. Locate the line that says **Clone this repository**.
6. Highlight and copy the `git` command line provided.

7. Switch to your terminal window and navigate to your new `repos` directory.
8. Paste the command you copied from Bitbucket onto the command line and press **Return**.
   Git will ask you for the repo password. This is password you entered when you created your Bitbucket account.
9. Enter your password.
   Git creates the repository but warns you that you have cloned an empty repository. You already knew that your repository was empty right? Recall that you had no commits yet in it.
10. List the contents of your repos directory and you can see your`bb101repo`.
    At this point, your terminal window should look something like this:

```
$ cd ~/repos
$ git clone https://newuserme@bitbucket.org/newuserme/bb101repo.git
Cloning into 'bb101repo'...
Password
warning: You appear to have cloned an empty repository.
$ls
bb101repo
```

Step 2. Explore your repo and fix a problem

Git said you had cloned and empty repository. This is true, your repository is empty but the directory that git created is not entirely empty.
List the contents of your *local repo* – including the hidden files. A local repo is the copy of a repo you have on your local system. After listing the contents of your local repo, you should see something like this:

```
$ ls -al bb101repo/
total 0
drwxr-xr-x  3 manthony  staff  102 Dec 14 10:50 .
drwxr-xr-x  3 manthony  staff  102 Dec 14 10:50 ..
drwxr-xr-x  9 manthony  staff  306 Dec 14 10:50 .git
$
```

The `.git` directory contains special files and directories used by the Git system. For now, you should be aware that it is there.

Notice that Git went ahead and named your repo's directory with the same name as you did when you created it. You could have named the directory something totally different. For example, you could have named the directory something that indicated what you were going to do with it. For example, you could have named it `bb101repo-prod`. In fact, it is good practice not to simply use the repo name when you clone but indicate what you are doing with the clone. Why don't you fix that right now:

1. Remove the repo you just created.

   ```
   rm -irf bb101repo/
   ```

2. Reissue the clone command but this time give Git a name that indicates what you are doing.
   For example, you might want to call the clone `bb101repo-practice`:

   ```
   $ git clone https://newuserme@bitbucket.org/newuserme/bb101repo.git bb101repo-practice
   ```

   Again, Git will tell you are cloning an empty repo.

3. List your `~/repos` directory, you should see something similar to the following:

   ```
   $ ls ~/repos
   bb101repo-practice
   ```

Great. Now you are reading to begin adding content to your repo.

Step 2. Create a README and add it to your Bitbucket repo

Bitbucket let's you set a repo's description but you may want to provide detailed information or instructions about your repo. You do this in a README file that is at the root of your repo's code base. There are a several formats you can use to create a README. This tutorial shows you how to create a plaintext README in your local repo and push it back to your Bitbucket repo.

1. Go to your terminal window and navigate to the top level of your local repo.

   ```
   cd ~/repos/bb101repo-practice/
   ```

2. Using your favorite editor, create a `README` file with the following content:

   ```
   Welcome to My First Repo
   -----------------------------
   This repo is a practice repo I am using to learn Bitbucket.
   ```

3. When you are done, list the contents of your local repo.
   You should see something similar to the following:

   ```
   $ ls -al
   total 8
   drwxr-xr-x  4 manthony  staff  136 Dec 14 11:50 .
   drwxr-xr-x  3 manthony  staff  102 Dec 14 11:30 ..
   drwxr-xr-x  9 manthony  staff  306 Dec 14 11:30 .git
   -rw-r--r--  1 manthony  staff  117 Dec 14 11:50 README
   ```

4. Go ahead and list the status of your local repo.
   The `git status` command tells you about how your project is progressing in comparison to your Bitbucket repo. You should see something like this:

```
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#    README
nothing added to commit but untracked files present (use "git add" to track)
```

You can see that git is aware that you created a file in your local repository. The status output also shows you the next step, the add.

5. Add your new README file to a pending commit.

```
git add README
```

What happens if you run the status command now? Git sees that you have a new file in your local repo and that you can potentially commit it.

6. Commit all the changes you added.
   Right now, the only change you added to your local repo was the new README. When you commit you see this:

```
$ git commit -m "adding repo instructions"
[master (root-commit) 12ad229] adding repo instructions
 1 files changed, 3 insertions(+), 0 deletions(-)
 create mode 100644 README
```

Up until this point, everything you have done is only visible to you. There is nothing at all in your Bitbucket repository that shows your work.

7. Open your Bitbucket bb101repo **Overview** page in your browser.
   You should see that the **Overview** looks exactly as it did when you started.

8. Go back to your terminal window and push your committed changes back to your Bitbucket repository.
   You should see something similar to the following:

```
$ git push -u origin master
Password:
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 303 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: bb/acl: newuserme is allowed. accepted payload.
To https://newuserme@bitbucket.org/newuserme/bb101repo.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

9. Open your Bitbucket **bb101repo Overview** page in your browser.

   You should see two changes, you now have a commit on your repo and the README text is showing up.

Bitbucket traps a lot of information with your commit and shows it to you. You can see that the **Author** column shows the value you used when you configured the DVCS – in Git's case the `~/.gitconfig` file.

10. Select your repo's **Source** tab.
    You should see that you have a single source file in your repo, the `README` file you just added.

Next

Now, you have finished the simplest workflow between Bitbucket and a DVCS system. You created a local repository, added a new file to it, and pushed those changes to the Bitbucket repository. At this point you may have a few thoughts in your head such as:

- Hey, I don't consider a README file to be source code.
- Aren't other people supposed to help me with my repo?
- This Git stuff is fine but I want to use Mercurial.

The next page this tutorial help resolve some of these thoughts because you will use Mercurial to add code to another user's repository.

Fork a Repo, Compare Code, and Create a Pull Request

So far, you've worked in a repo you own and you've been the only person working in it.   You can only fully experience the power of a DVCS hosted repository by working with others. In this page, you work with a repo you don't own. You'll make a change to a code file, not just a README. You'll use bitbucket's comparison features to compare your repo with the original.

The tutorial examples so far worked exclusively with Git repos using `git` commands. In this section, you'll work with a Mercurial repo. If you start working extensively in the host DVCS community, you'll most likely find yourself working with multiple DVC systems.   You may even find yourself working with other hosted products, such as GitHub or Kiln. It is good experience to see the workflow you'd use in these types of situations.

> ℹ The examples on this page are written for Mercurial using TortoiseHg on Microsoft Windows 7. If you want to work in Mac OSX or Linux, see the instructions on this page.

Backgrounder

When you work with another user's public bitbucket repo, typically you have read access to the code but not write access. This means that you can use a `clone` command to copy the repo but you cannot `push` changes to it. Instead, you use the bitbucket interface to *fork* a repo. bitbucket hosts private or public repos. Anyone can fork a public repo. You can fork a private repo where you have permissions.

Forking a repository creates a new repository under your account. This new repository is a copy of the original repo and is called a *fork*. Even though you have a fork, you can still get a change into the original repository, to do this you:

- clone the forked repo from bitbucket to your local system
- make changes to the local repo
- push the changes to your fork on bitbucket
- issue a *pull request* against the hosted repo you forked from
- wait for the repo owner to accept or reject your pull request

If a repo owner accepts a pull request, bitbucket pulls your code changes into the original repo and merges them. bitbucket recommends you work with forks and pull requests even if the repo owner gives you write access to a public repository. While a pull is a DVCS concept, a "pull requests" and forks are concepts used only by repository hosting services — like bitbucket and GitHub. You won't find fork and pull requests in any Git or Mercurial workflow that doesn't involve a repository hosting service.

Step 1. Fork another user's repo

In this example you'll fork a public repository belonging to a user called `tutorials`.

1. Log into bitbucket.
2. Search for and then click through to the `tutorials.bitbucket.org` repo.
3. Choose the **fork** option from the repository menu.
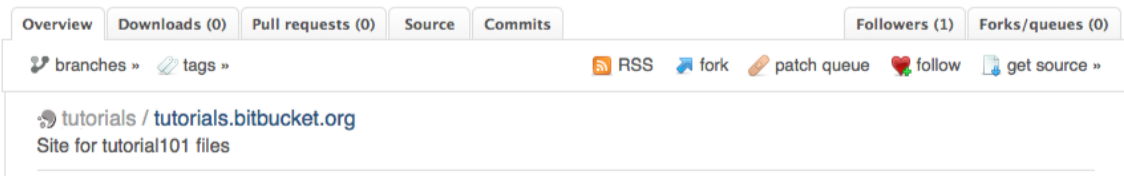
The system displays the fork page.
4. Change the **Name** for example, to `myquotefork`.
5. Enter a **Description** that you you think is appropriate.
   There are several options for forking. For now, just leave all the remaining options at their defaults.
6. Press **Fork repository**.

Step 2. Clone your fork

1. Start the TortoiseHG Workbench.
2. Choose **View > Show Repository Registry**.
3. Choose **File > Clone New Repository**.
4. Enter location of your forked repo in the **Source** field.
5. Enter a destination on your system for your local repo, for example:
   `C:\Users\yourusername\repos\myquotefork`
   When you are done the dialog will look similar to the following:
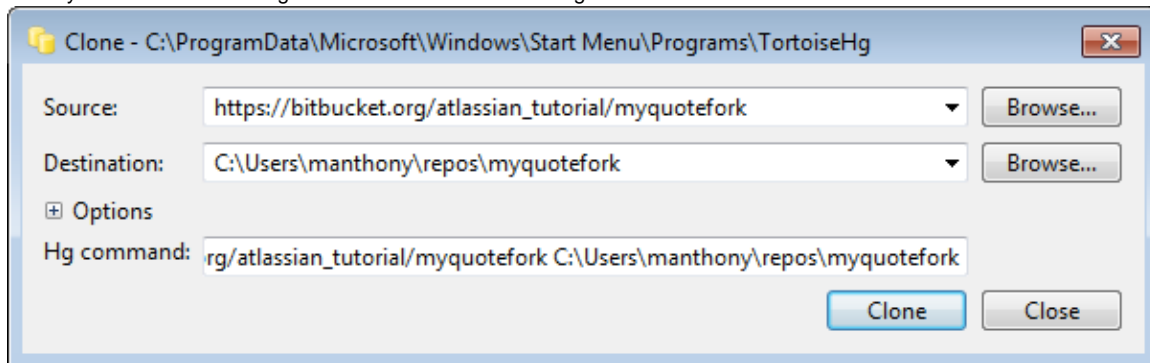
6. Press **Clone**.
   The system clones the repository and displays it in the registry list.

Step 3. Make a change to the repository source

This repository contains a website which, as of this writing, has a single `index.html` file. bitbucket allows you to host a website in a Mercurial repository. To see the hosted website, go to https://tutorials.bitbucket.org – you may encounter an Untrusted Connection message. Go ahead through to the site. You'll see that the site contains a single page that lists favorite quotes from "bitbuckians" which is just a writer-invented word for users of bitbucket. Now, it is your turn to record your favorite comedic quote...or just a favorite quote. Do the following to contribute to this repository:

1. Use Google or some other search engine to locate your favorite quote.
2. In the TortoiseHG Workbench, select the `index.html` file.
3. Right-click to display the context menu.
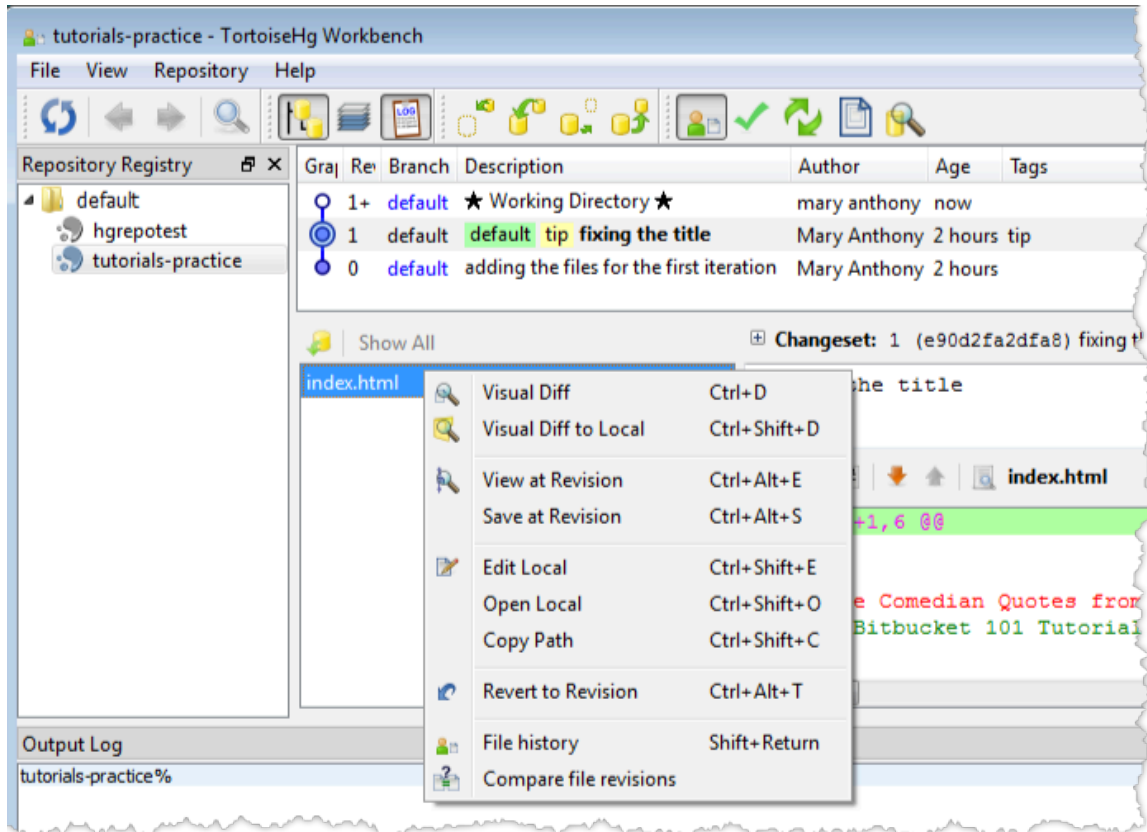
4. Choose **Edit Local**.
   The file is an HTML file.
5. Go ahead and insert an image of the person you are quoting and their quote right above the `</table>` tag in the file.
   If you are not sure what to do, you can copy and modify another user's entry. Most users used the following suggested structure when adding a quote:

```
<tr>
<td>
 <img src="PATH_TO_IMAGE"/>
</td>
<td>
 <blockquote>QUOTE_YOU_WANT_TO_ADD</blockquote>
</td>
</tr>
```

   You can actually do whatever you like here (ok, within reason you can).   For example, one user cleaned up our HTML — thanks Jon! And another added some nifty CSS – thanks Andrei!

6. Close and save the file.
7. Refresh the **Working Directory**.
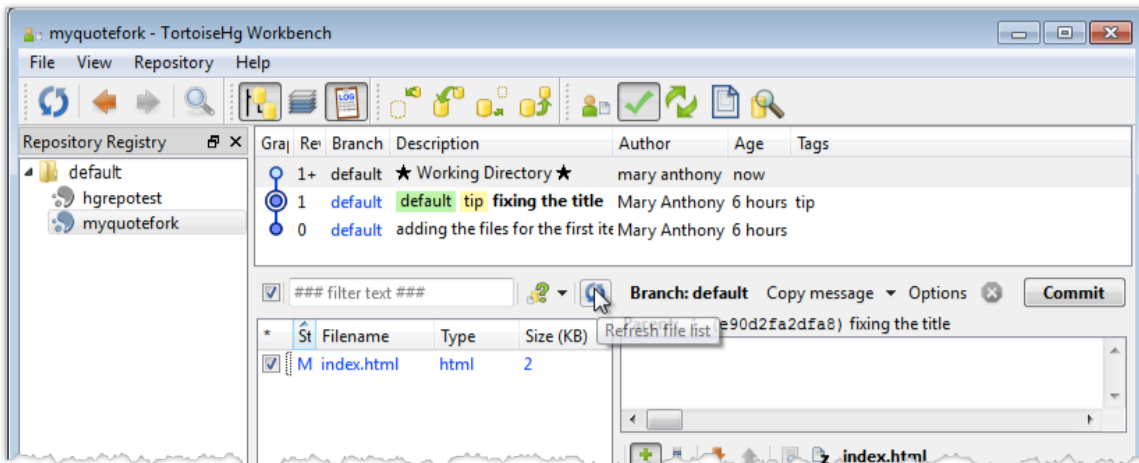
8. Enter a commit message in the space provided.
9. Press the **Commit** icon (check symbol) to commit your changes.
10. Press the **Push outgoing changes to selected URL** icon.

The system prompts you to confirm your action.
11. Press **Yes**.
    The system pushes your changes to the forked repo.

Step 4 . Compare your fork to the original

During the time you were working on your fork, the original repository you forked from may have changed. At this point, you can check that and decide if you need to adjust your fork accordingly. Log into bitbucket and navigate to your `myquotefork` repository.   Forked repos have a special widget that lets you compare your fork work to the original or to send a pull request. The **compare fork** and **send pull request** buttons toggle between two specialized views available only in forked repos. Press **compare fork** to reveal the compare view.
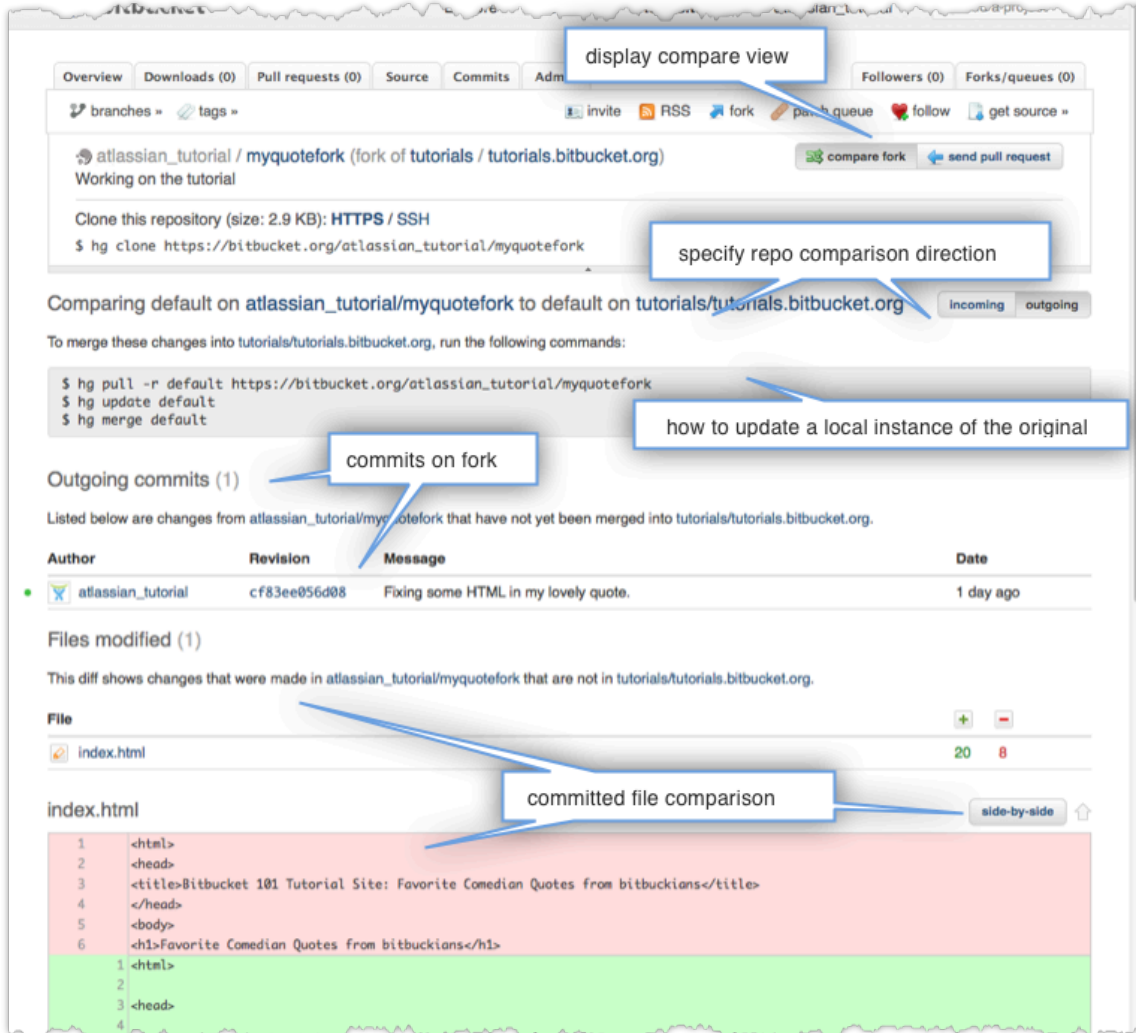
This view allows you to look at your forked repo in comparison to the original repo. You can look at it from two "directions." You can compare your repo against **incoming** changes from the original. Or, you can compare **outgoing** changes from your fork to the original.

Keep in mind that the outgoing changes can be from multiple people. How? Well if you have shared your fork repo with others. (You'll learn more about how to do this later.) Right now, though, you should be the only person on the repo so only your changes appear.

This view includes helpful Mercurial commands. You can merge your fork into another repo — for example a local copy you may have of the original repo. If you merge locally, you can test your changes before making a pull request through bitbucket.

The commits section displays the commits pushed from a local repo to the fork in bitbucket. If there are multiple commits, you see their cumulative changes by file in the **Files modified** section. You can use the **side-by-side** button to see the changes displayed in that format.

To see the contents of a specific commit in isolation, select a **Revision** and the system takes you to the **Commits** page.

Step 4. Create a pull request

If you haven't already done so, log into bitbucket and navigate to your `myquotefork` repository. Then, do the following:

1. Press **send pull request**.

The system displays the request form.
2. Complete the form.
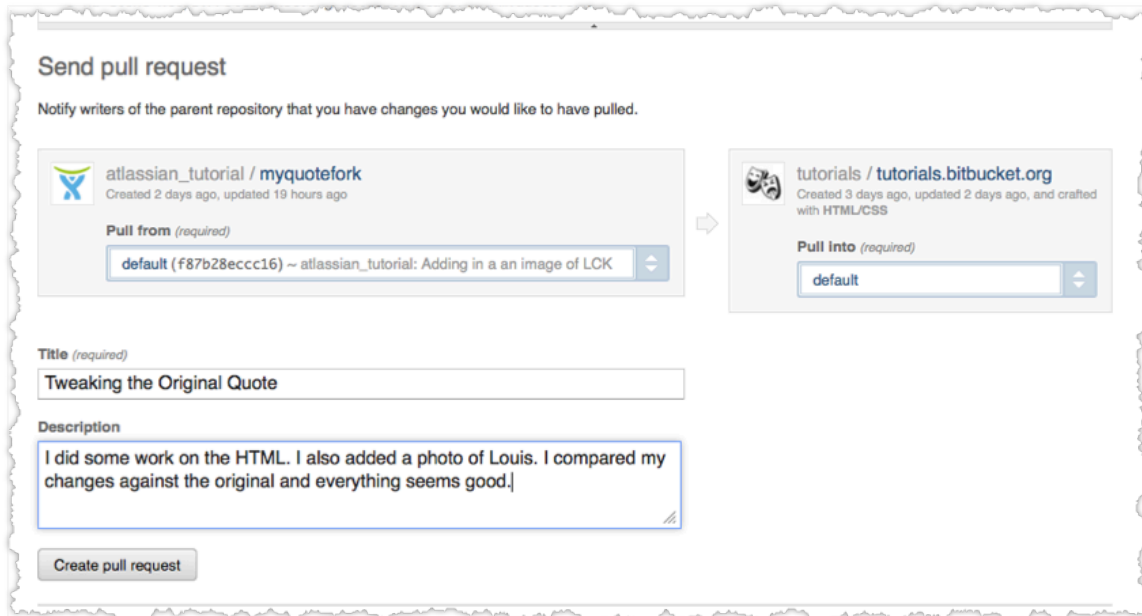
3. Press **create pull request**.
   The system opens your latest request on the **Pull Request** page *of the original repo*. To see the list of all the pull requests against this repo, click the **Pull Request** tab. The system also sent a notification email to your account Inbox.
4. Go ahead and open your account inbox ***username* > Inbox**.
5. Locate and review the pull request email (it should be near the top).

Step 6. Learn what happens to your pull request

Accepting the pull request is not something you do it is something done by the repo owner. It is the next step though. When the original repo owner logs into his account, his **Newsfeed** shows your pull request. It also shows your fork from a few days earlier. For example:

The repo owner can click on the request and see your full request. An owner can reject or accept a request.

⚠ There are a couple of things you should know. You can't delete a pull request once you've created one; the receiver can't either. Your request is either accepted or rejected. Also, if you delete your fork after you make the request, your request can only be rejected because the repo to pull from is gone.

You'll get an email when your pull request is accepted or rejected. For now, continue to the next step.

Next

That was intense. Maybe. Depends on your daily life. In the next section, you learn ways to be social on bitbucket which includes adding users to your repository.

Add Users, Set Permissions, and Review Account Plans

Forking a repo is the perhaps the best way to work with others on bitbucket. There are other ways to interact and collaborate. For example, you can follow another user or add a user to your repository. You can do the procedures on this page in any order you like.

Show an interest in others

There are a lot of ways to discover new people and code. Log into bitbucket and click **Explore**.

Areas to explore	This shows you
Trending repositories	Public repositories with a lot of activity such as commits, followers, or forks. You can choose what period of time to calculate these over.
Featured repositories	A revolving list of public repositories in bitbucket. We choose these for you.
Blogospherical disturbances	The latest tweets about bitbucket and/or the repositories it hosts. A search service compiles this list. Use the links to navigate to a Twitter feed or to the post.

When you see a ***username / repository*** combination in trending or featured repositories, these are links you can click. Clicking on a username takes you to the user's overview page. From that page you can:

* get more information about a user such as location or website
* browse through a user's public repositories and latest commit activity
* send a user a message
* follow the user's future activities (the heart icon)

Clicking on a repository takes you to the repo overview. Here you can follow the repo (heart icon!) or explore the project in greater depth.

Add users to your repository

If you own a public repository, any bitbucket user can fork that repo, make changes, and send a fork request to you. However, you may want keep your repository private. In that case, you can add users you know or invite users you think may want to contribute to view or work with your repo. For both public and private repos, you can give users the permission to push changes directly to your repo or more, administrate your repo. To do this, you add a user and set their permissions.

1. Go to the **Admin** page for you `bb101repo` repository.
2. Click on **Access Management**.
3. Start typing a username in the field provided.
   The system provides drops down a list of possible users that match.
   If you don't know a username you want to add, try adding the `tutorials` user or `atlassian_tutorial`.
4. Select a matching name from the dropdown.
5. Select a permission:

Level	This permissions allows a user to do the following:
read	View the repository contents. All public repositories grant all bitbucket users read permissions automatically.
write	Contribute to the repository by pushing changes directly from a repository on a local machine.
admin	Do everything a repository owner can do, except delete the repository. This means administrators can: • Change repository settings. • Add, change, and remove user permissions. • Give other users administrator access.

   The system adds the user to the **Users** list with the permissions you selected. It also sends an email to the user informing them about the add.

Notification: You have hereby been granted write access to bb101repo.

BitBucket/Notices   x

Bitbucket notifications-noreply@bitbucket.org   12:08 (13 minutes ago)

to me

You have received a notification from newuserme.

You may now access the repository source on:

https://bitbucket.org/newuserme/bb101repo/overview
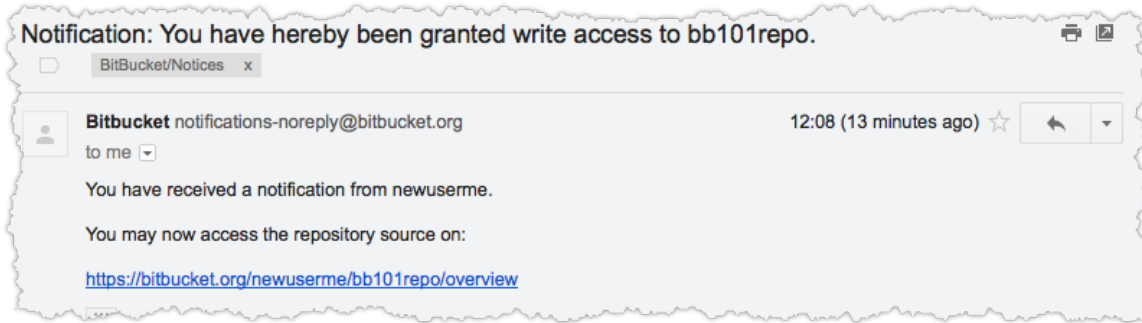
If you make a mistake, delete the user by pressing 🚫 and try again.

Verify Plans and users

bitbucket allows everyone with a free account an unlimited number of public and private repositories. You can grant as many users as you want access to your public repos. bitbucket plans restrict the number of users *with access to your private repositories.* Users with permissions such as write or admin on public repos don't count. With a free plan, you get up to 5 users, including yourself, with access to your private repos.

After adding users to your account, you should review your plan and note which users that "count" toward it. To view your plan, do the following:

1. Choose *username* **> Account**.
2. Choose **Plans and billing** from the left hand panel (or scroll to the page bottom).
   The system displays your current plan status:

Plans and billing

2 / 5 private users   [Change plan]

The number of users (including you) with any level
of access to one or more of your private repos.

Access via groups

Direct access

👤 newuserme

▶ 🏆 atlassian_tutorial   🚫

[   ]   [Read ▾]   [Add private user]

Grant this user access to my current private repositories.
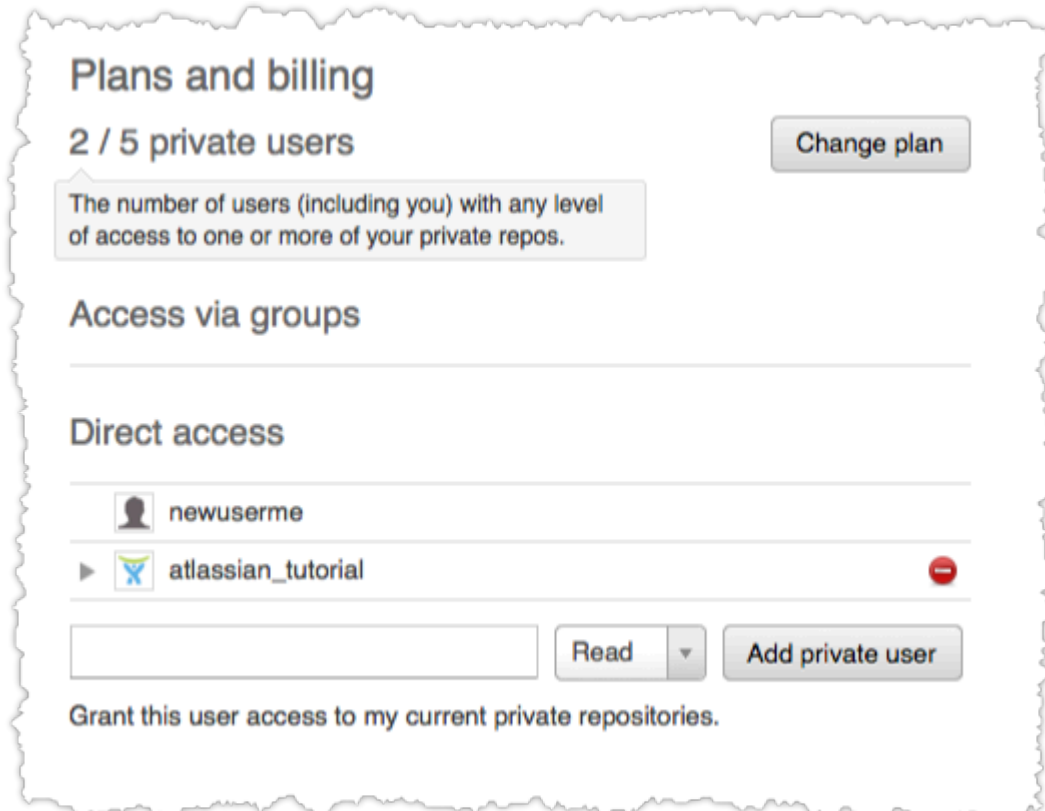
In this example, two users have access **newuserme** (the current account) and **atlassian_tutorial**. The plan shows that the **newuserme** account is using 2 plan users – both the account user and the **atlassian_tutorial** have access to the `bb101repo` which is private.

How would your accounts **Plans and billing** appear if you change `bb101repo` from private to public? Go ahead and try it.

Next

Beyond code contributions, users can also contribute to your repos by writing documentation in a wiki or making comments through an issue tracker. In the next section, you enable a wiki and an issue tracker on the `bb101repo`.

Set up a Wiki and an Issue Tracker

When you add a repository to bitbucket, you can also enable a wiki and an issue tracker for that repository. The wiki is a simple place to keep

documents. The issue tracker is the place to track your projects feature requests, bug reports and other project management tasks. The wiki and issue tracker do not depend on each other. You can choose to set up each one separately. This page leads you through the basics of using the wiki and issue tracker.

Step 1. Configure your wiki

To enable the wiki for your repository, log into bitbucket and do the following:

1. Navigate to your `bb101repo`.
2. Go to the repository **Admin** tab.
3. Click **Wiki settings**.
   The system opens the Wiki page. A private wiki is visible only to people who give permission to see it. A public wiki anyone can view, edit, or create pages.
4. Click **Private wiki**.
   The system enables the **Wiki** tab for your repository.

Step 2. Update wiki pages

A bitbucket wiki is a repository like any other. You can clone it and push changes to it like any other repository. To create content, the wiki uses Creole markup.

1. Click the **Wiki** tab for your `bb101repo`.
   By default the system displays the wiki **Home** page.
2. Click **Edit**.
3. Make your changes to the page content.
   You may want to change the `Welcome` heading to say `Welcome to the bb101repo`.
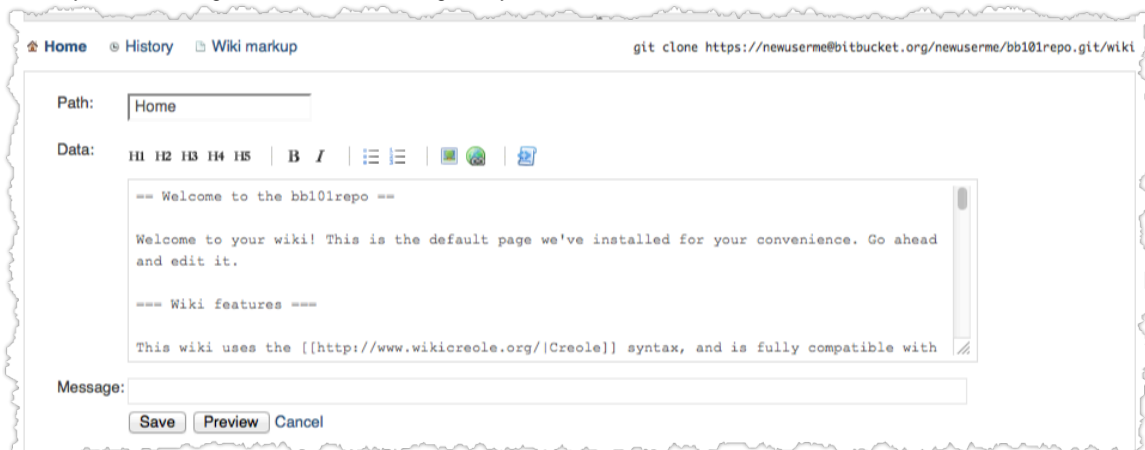
   If you want to do something more elaborate but you aren't sure how, take the **Wiki markup** link to view the help. The bitbucket wiki uses the Creole syntax.
4. Enter a comment in the **Message** text box.
   This is a commit message. What you enter here will appear in the page history above the relevant commit entry.
5. Click **Save**.
   The system displays your home page with the new heading.

Step 3. Configure your issue tracker

To enable the wiki for your `bb101repo`, log into bitbucket and do the following:

1. Navigate to a repository.
2. Go to the repository **Admin** tab.
3. Click **Issue tracker settings**.
   The system opens the Wiki page. A private wiki is visible only to people who give permission to see it. A public wiki anyone can view, edit, or create pages.
4. Click **Private tracker**.
   The system enables the **Issue tracker** tab for your repository. You will see there are now a number of settings available.
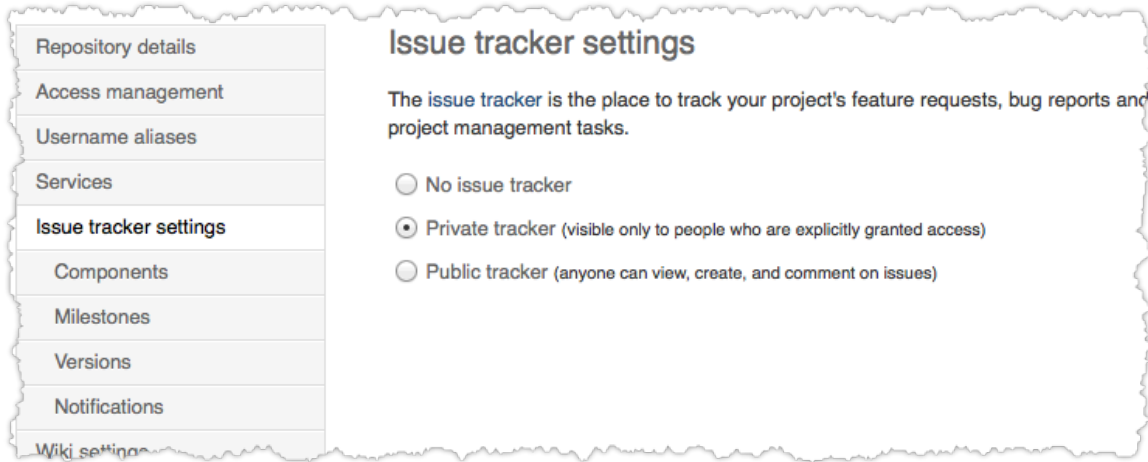
5. Click **Components** and configure the setting by
   You can use components to group issues within a project into logical groups.
6. Add both a **site design** and **performance** component.
7. Ignore the **Milestones** setting.
   A milestone is a subset of a version. It is a point that a development team works towards. Like all settings **Milestones** is optional. You won't need it for this tutorial so leave it unset.
8. Click **Versions** and configure the setting by adding a 1.0 value.
   A version helps you schedule and organize your releases and track the release that is affected by a bug.
9. Click **Notifications** and configure the setting by adding a back up email for yourself.
   These are email addresses that receive notification when a user creates an issue.

Step 4. Create an issue

Creating bug reports, improvement requests and tasks is as simple as creating an issue of the appropriate type.

1. Go to the repository's **Issues** tab.
2. Click **Create Issue**.
3. Fill in the fields on the issue form.
   The system displays the create issue panel:
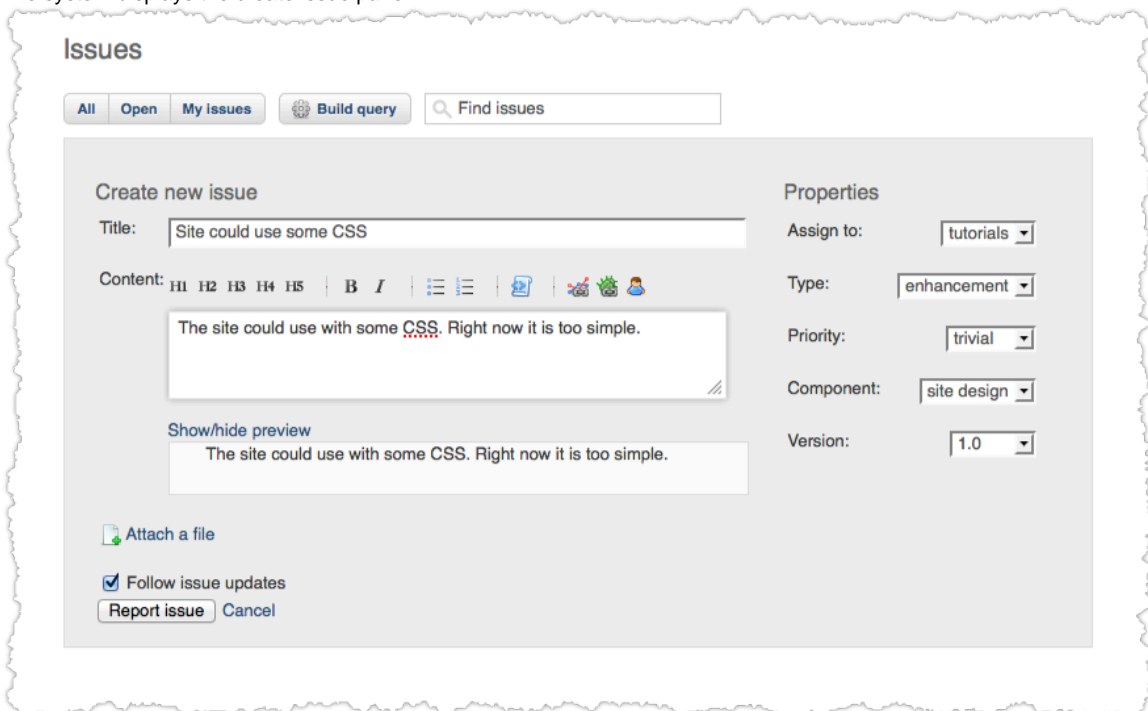
Notice that this issue tracker is not using a milestone value.

4. Click the **Report Issue** button to create the issue.

Next Steps

Up until this point, you've used HTTPS to access your bitbucket repos. HTTPS access requires that you give a username and password for each operation. If you are doing a lot of work, this can get annoying. In the next section, you learn how to use secure shell (SSH) to access your bitbucket repos.

Set up SSH for Git

Up until this point, you have been using the secure hypertext transport protocol (HTTPS) to communicate between your local system and bitbucket. When you use HTTPS, you need to authenticate (supply a username and password) each time you take an action that communicates with the bitbucket server. You can specify the username in the DVCS configuration file; you don't want to store your password there though where anyone can see it. So, this means you must manually type a password when you use HTTPS with your local repo. Who wants to do that? This page shows you how to use secure shell (SSH) to communicate with the bitbucket server and avoid having to manually type a password.

> ℹ️ This page shows you how to set up and use a *single default SSH identity* on Windows for a Git repo using GitBash. In the next page, you set up SSH for a Mercurial repo on Windows with TortoiseHg. If you are working on Mac OSX or Linux, a single set of instructions shows you how to setup and identity for either Git or Mercurial in these environments.

Finally, setting up an SSH identity can be prone to error. Allow yourself some time, perhaps as much as an hour depending on your experience, to complete this page. You can even skip this whole page and continue to use HTTPS if you want.

Step 1. Read a quick overview of SSH concepts

To use SSH with bitbucket, you create an SSH identity. An identity consists of a private and a public key which together are a key pair. The private key resides on your local computer and the public you upload to your bitbucket account. Once you upload a public key to your account, you can use SSH to connect with repos you own and repos owned by others, provided those other owners give your account permissions. By setting up SSH between your local system and the bitbucket server, your system uses the key pair to automate authentication; you won't need to enter your password each time you interact with your bitbucket repo.

There are a few important concepts you need when working with SSH identities and bitbucket

- You cannot reuse an identity's public key across accounts. If you have multiple bitbucket accounts, you must create multiple identities and upload their corresponding public keys to each individual account.
- You *can* associate multiple identities with a bitbucket account. You would create multiple identities for the same account if, for example, you access a repo from a work computer and a home computer.   You might create multiple identities if you wanted to execute DVCS actions on a repo with a script – the script would use a public key with an empty passphrase allowing it to run without human intervention.
- RSA (R. **R**ivest, A. **S**hamir, L. **A**dleman are the originators) and digital signature algorithm (DSA) are key encryption algorithms. bitbucket supports both types of algorithms. You should create identities using whichever encryption method is most comfortable and available to you.

Step 2. Check if you have existing default Identity

The Git Bash shell comes with an SSH client. Do the following to verify your installation:

1. Double-click the Git Bash icon to start a terminal session.
2. Enter the following command to verify the SSH client is available:

```
manthony@MANTHONY-PC ~
$ ssh -v
OpenSSH_4.6p1, OpenSSL 0.9.8e 23 Feb 2007
usage: ssh [-1246AaCfgkMNnqsTtVvXxY] [-b bind_address] [-c cipher_spec]
[-D [bind_address:]port] [-e escape_char] [-F configfile]
[-i identity_file] [-L [bind_address:]port:host:hostport]
[-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
[-R [bind_address:]port:host:hostport] [-S ctl_path]
[-w local_tun[:remote_tun]] [user@]hostname [command]
```

3. If you have `ssh` installed, if you do, go to the next step.
   If you don't have `ssh` installed, install it now with you package manager.

4. List the contents of your `~/.ssh` directory.
   If you have not used SSH on Bash you might see something like this:

```
manthony@MANTHONY-PC ~
$ ls -a ~/.ssh
ls: /c/Users/manthony/.ssh: No such file or directory
```

If you have a default identity already, you'll see two `id_*` files:

```
manthony@MANTHONY-PC ~
$ ls -a ~/.ssh
.         ..          id_rsa         id_rsa.pub     known_hosts
```

In this case, the default identity used RSA encryption (`id_rsa.pub`). If want to use an existing default identity for bitbucket account, skip the next section and go to install your public key.

Step 3. Set up your default identity

By default, the system adds keys for all identities to the `/Users/yourname/.ssh` directory. The following procedure creates a default identity.

1. Open a terminal in your local system.
2. Enter `ssh-keygen` at the command line.
   The command prompts you for a file to save the key in.
3. Accept the default location.
4. Enter and renter a passphrase when prompted.
   Unless you need a key for a process such as script, you should always provide a passphrase. The command creates your default identity with its public and private keys. The whole interaction will look similar to the following:

```
manthony@MANTHONY-PC ~
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/manthony/.ssh/id_rsa):
Created directory '/c/Users/manthony/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/manthony/.ssh/id_rsa.
Your public key has been saved in /c/Users/manthony/.ssh/id_rsa.pub.
The key fingerprint is:
e7:94:d1:a3:02:ee:38:6e:a4:5e:26:a3:a9:f4:95:d4 manthony@MANTHONY-PC
manthony@MANTHONY-PC ~
$
```

5. List the contents of `~/.ssh` to view the key files.

Step 4. Create a config file

1. Using your favorite text editor, edit an existing (or create a new) `~/.ssh/config` file.
2. Add an entry to the configuration file using the following format:

```
Host bitbucket.org
 User accountname
 IdentityFile ~/.ssh/privatekeyfile
```

The indentation is important, so make sure you indent the 2nd and 3rd lines.

When you are done your configuration looks similar to the following:

```
Host bitbucket.org
 User newuserme
 IdentityFile ~/.ssh/id_rsa
```

3. Save and close the file.
4. Restart the GitBash terminal.
   When you start GitBash, it starts an agent running for you.
5. Confirm the `ssh-agent` is running with the `ps` command.
   You should see something similar to the following:

```
$ ps
      PID    PPID     PGID    WINPID  TTY  UID    STIME COMMAND
     5192       1     5192      5192    ?  500 19:23:34 /bin/ssh-agent
     5840       1     5840      5840  con  500 08:38:20 /bin/sh
     6116    5840     6116      1336  con  500 08:38:22 /bin/ps
```

If for some reason the agent isn't running, start it by entering `eval ssh-agent` at the command line. You should only be running a single instance of `ssh-agent`. If you have multiple instances running, use the `kill PID` command to stop each of them. Then, restart a single instance.

Step 5. Install the public key on your bitbucket account

1. Open a browser and log into bitbucket.
2. Choose *Username* > **Account** from the menu bar.
   The system displays the **Account settings** page.
3. Locate the **SSH keys** section and the **Add key** field.
4. In your terminal window, `cat` the contents of the public key file.
   For example:

   ```
   cat ~/.ssh/id_rsa.pub
   ```

5. Select and copy the key output in the clipboard.
   If you have problems with copy and paste, you can open the file directly with Notepad. Select the contents of the file (just avoid selecting the end-of-file character).

6. Paste the captured key into the field provided and press **Add key**.

   qp7t2zOjWEXpQ== manthony@MANTHONY-PC   Add key

   The system adds the key to your account.

Using the Choose key button

If your browser has the **Choose key** button, you can do this instead:

1. In the **SSH keys** section, click **Choose key**.
   The system displays the **File Upload** dialog.
2. Navigate to your `C:\Users\username\.ssh` directory.
   This is a hidden directory. If hidden directories are invisible, in Mac OSX press **command+shift+.** (period) . In Windows set your file folder preferences to view hidden files and try again.
3. Locate the public key file (`filename.pub`) and click **Open**.
   The **Choose key** button changes to **Upload key** and the file name appears next to it.

4. Press **Upload key**.

Step 6. Configure your repo to use the SSH protocol

The URL you use for a repo depends on which protocol you are using, HTTPS and SSH. The bitbucket repository **Overview** page has a quick way for you to see the one for your `bb101repo` repo. On the repo's **Overview** page look for the **Clone this repository** line.

Experiment for a moment, click back and forth between the **SSH** and the **HTTPS** protocol links to see how the URLs differ. The table below shows the format for each DVCS based on protocol.

	SSH format	**HTTPS format**
Mercurial	`ssh://hg@bitbucket.org/`*accountname*`/`*reponame*`/`	`https://`*accountname*`@bitbucket.org/`*accountname*`/`*reponame*

Git	`git@bitbucket.org:`*accountname*`/`*reponame*`.git`
	
	or
	
	`ssh://git@bitbucket.org/`*accountname*`/`*reponame*`.git`
		`https://`*accountname*`@bitbucket.org/`*accountname*`/`*reponame*`.git`

Go to terminal on your local system and navigate to your `bb101repo-pratice` repo. Then, do the following:

1. View your current repo configuration.
   You should see something similar to the following:

```
manthony@MANTHONY-PC ~
$  cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
    ignorecase = true
[remote "origin"]
    fetch = +refs/heads/*:refs/remotes/origin/*
    url = https://newuserme@bitbucket.org/newuserme/bb101repo.git
[branch "master"]
    remote = origin
    merge = refs/heads/master
```

   As you can see, the `url` is using the HTTPS protocol. There are a number of ways to change this value, the easiest way is just to edit the repo's configuration file.

2. Edit the `~/repos/bb101repo-pratice/.git/config` file with your favorite editor.
3. Change the `url` value to use the SSH format for that repo.
   When you are done you should the `origin` section should contain something similar to the following:

```
[remote "origin"]
    fetch = +refs/heads/*:refs/remotes/origin/*
    url = git@bitbucket.org:newuserme/bb101repo.git
```

4. Save your edits and close the file.

Step 7. Make a change under the new protocol

1. Edit the `README` file in your `bb101repo-practice` repo.
2. Add a new line to the file, for example:

```
Welcome to My First Repo
-------------------------------
This repo is a practice repo I am using to learn bitbucket.
You can access this repo with SSH or with HTTPS.
```

3. Save and close the file.
4. Add and then commit your change to your local repo.

```
git add README
git commit -m "making a change under the SSH protocol"
```

5. Push your changes.
   The system warns you that it is adding the bitbucket host to the list of known hosts.

```
manthony@MANTHONY-PC ~
$ git push
$ git push
Counting objects: 5, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 287 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: bb/acl: newuserme is allowed. accepted payload.
To git@bitbucket.org:newuserme/bb101repo.git
   056c29c..205e9a8  master -> master
```

6. Open the repo **Overview** in bitbucket to view your commit.

Next Steps

Now, you should set up SSH for Mercurial through the TortoiseHg Workbench. If you are a Mac OSX or Linux user, you should have already worked through one page that covers SSH for both Git and Mercurial.

Set up SSH for Mercurial

This page assumes you have already set up your Git repositories to use SSH. That page has some good information on it regarding how SSH works with bitbucket. If you haven't worked through that page, go back and do that now.

> ℹ This page shows you how to set up and use a *single default SSH identity* on Windows for a Mercurial repo. If you want set up for a Git repo on Windows using GitBash, there are instructions for Set up SSH for Git. Finally, if you are working on Mac OSX or Linux, a single set of instructions shows you how to setup and identity for both Git or Mercurial in these environments.

Finally, setting up an SSH identity can be prone to error. Allow yourself some time, perhaps as much as an hour depending on your experience, to complete this page. You can even skip this whole page and continue to use HTTPS if you want.

Step 1. Check if you have existing default identity

TortoiseHG relies on the PuTTYgen utility to generate an identity. If you are following along in this tutorial, you should have already installed PuTTYgen. If you used PuTTYgen in the past to generate an identity, you should have an `identity`.ppk file on your system. If you have an existing private key, you can skip Step 2 below and go onto Enable SSH compression for Mercurial. Otherwise, continue to the next step.

Step 2. Create your default identity

The following procedure creates a default identity.

1. Locate the `puttygen.exe` executaible in your system and double click the icon to start it.
   If you are following along with this tutorial, you installed PuTTYgen in `C:\Program Files\TortoiseHG`. The system opens the **PuTTY Key Generator** dialog.
2. Complete
3. Press **Generate**.
   Following the instructions to generate some randomness.
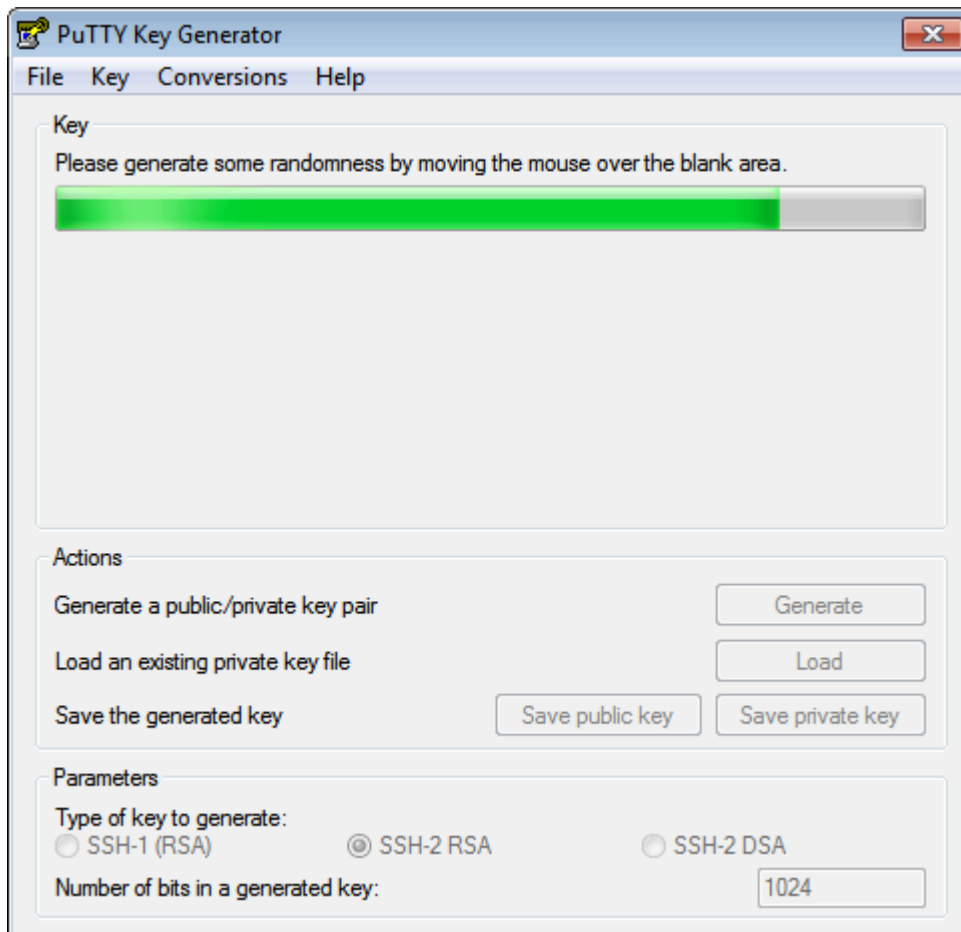
When the generation completes, the system displays the public key and a number of other fields.

4. Enter and confirm a key passphrase.
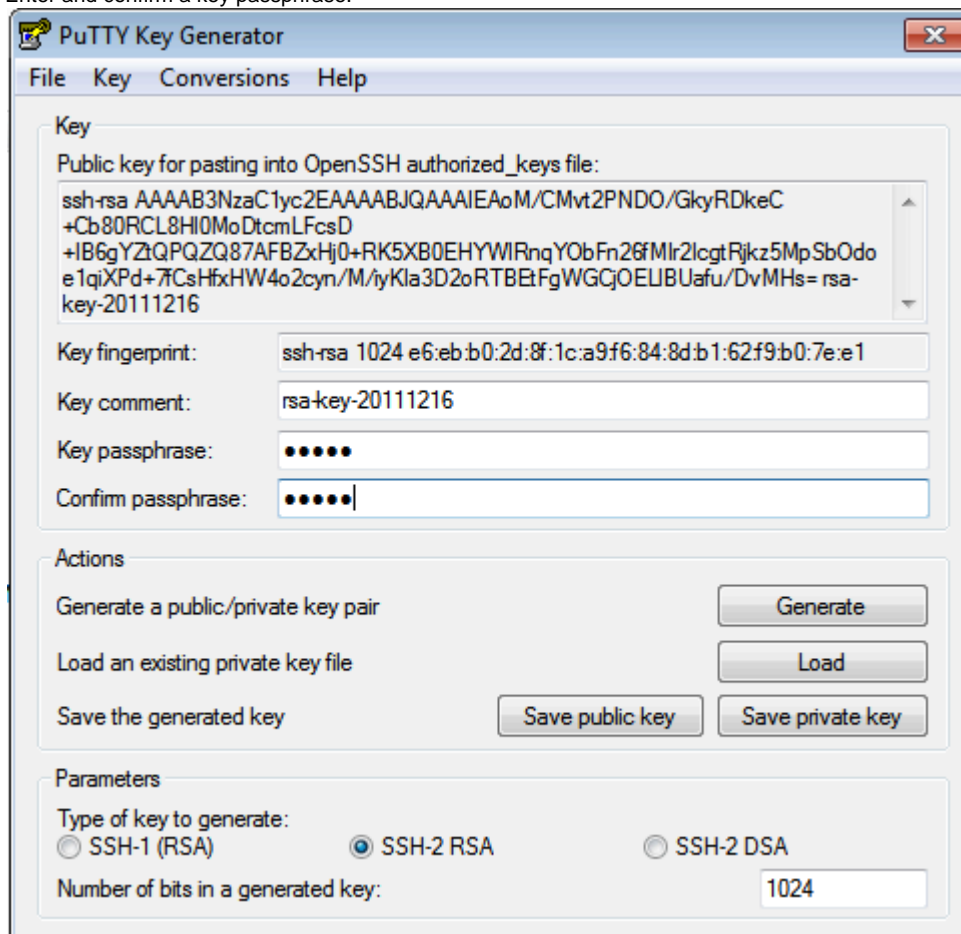
5. Press **Save private key**.
The system prompts you for a location to save the file and a file name. By convention, store your key files in a folder called

`C:\Users\`*`yourname`*`\.ssh.` and give it a `.ppk` extension.
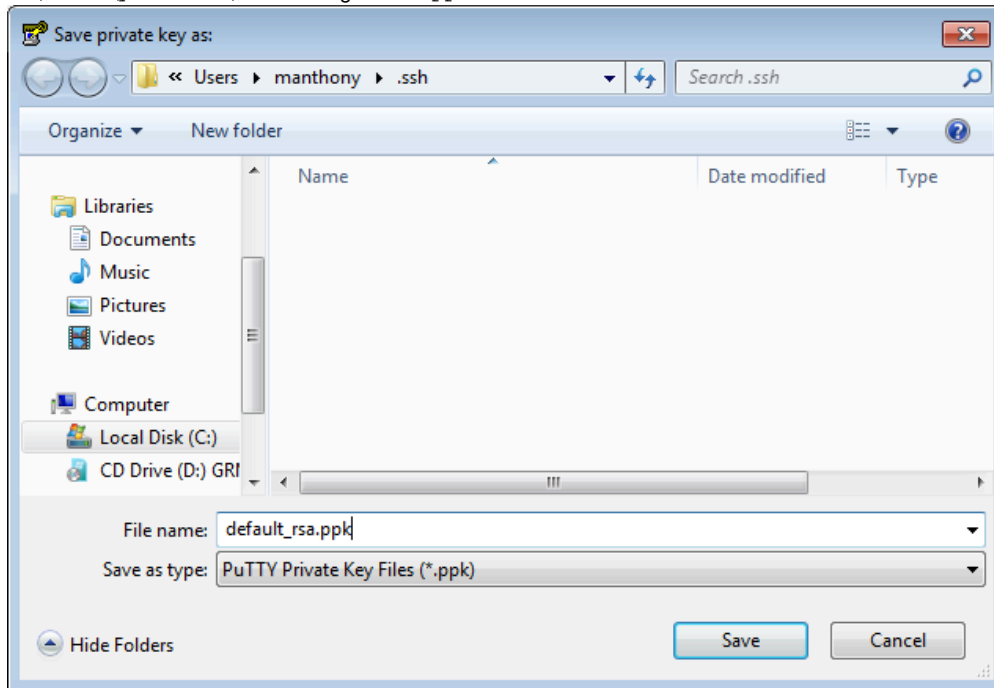
6. Go ahead and close PuTTYgen.

Step 3. Enable SSH compression for Mercurial

When sending or retrieving data using SSH, Git does compression for you. Mercurial does not automatically do compression. If you are using Mercurial, you should enable SSH compression as it can speed up things drastically, *in some cases.* To enable compression for Mercurial, do the following:

1. Start the TortoiseHg Workbench.
2. Select **File > Settings**.
3. Make sure you have the global settings tab selected.
4. Press **Edit File**.
5. Add the following line to the UI section:

```
ssh = "TortoisePlink.exe" -ssh -2 -batch -C
```

When you are done the file should look similar to the following:

```
[ui]
# Name data to appear in commits
username = Mary Anthony <manthony@atlassian.com>
ssh = "C:\Program Files\TortoiseHg\TortoisePlink.exe" -ssh -2 -batch -C
```

6. Press **Save** to store your settings and close the file.
7. Press **OK** to close the settings dialog.

Step 4. Start Pageant and install your private key

TortoiseHG comes with Pageant which is an SSH authentication agent. You load your keys into Pageant and it automatically authenticates you so you don't need to enter your passphrase. Do the following to load your keys:

1. Start Pageant by double clicking its icon.
   By default, TortoiseHG installs the Pageant in the `C:\Program Files\TortoiseHG` folder. When it is running, Pageant appears in your system tray:
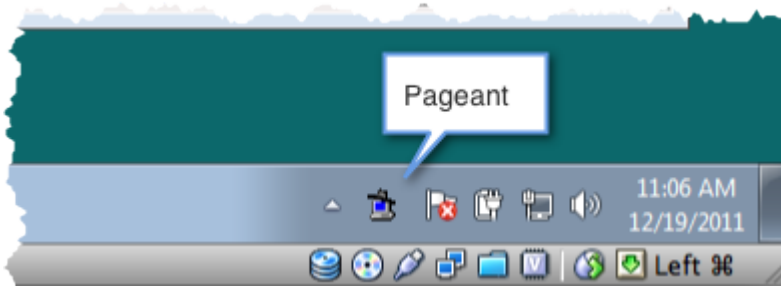
2. Double-click the Pageant icon to launch the **Pagent Key List** dialog.
3. Click the **Add Key** button.
   The system displays the **Select Private Key File** dialog.
4. Navigate to and open the default key you created previously.
5. Enter the passphrase when prompted:
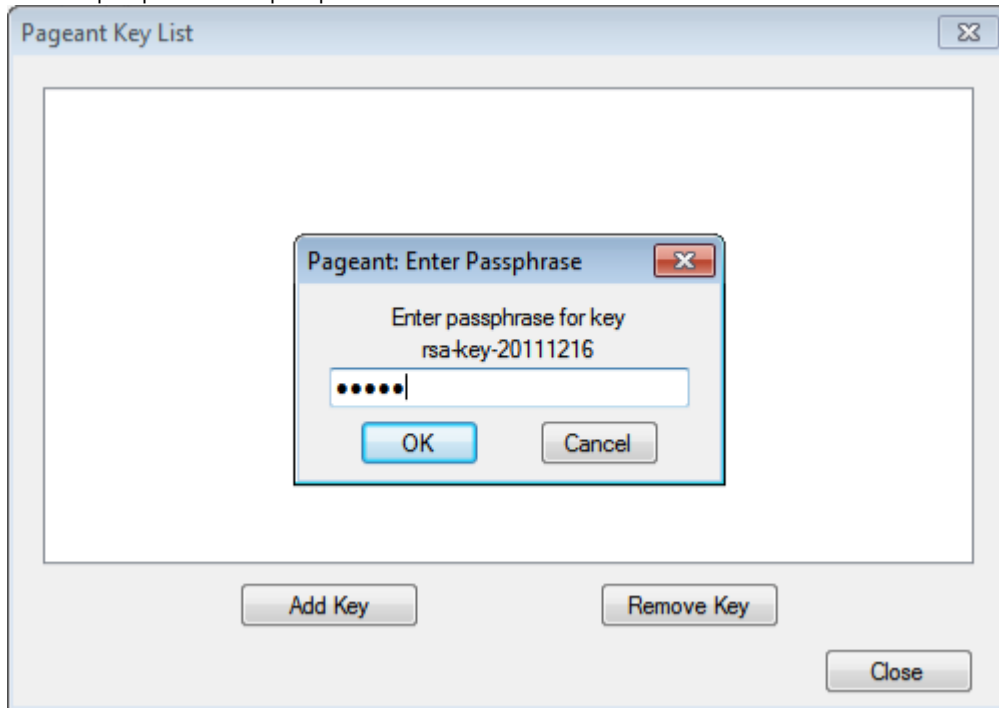
6. Press **OK**.
   Pageant shows your key in the running list.
7. Press **Close** to close the dialog.
   Pageant continues to run on your system.

Step 5. Install the public key on your bitbucket account

1. Open a browser and log in to bitbucket.
2. Choose *Username* **> Account** from the menu bar.
   The system displays the **Account settings** page.
3. Locate the **SSH keys** section and its **Add key** field.
   You will need to paste your public key into the Add key field.
4. Start the PuTTYgen program.
5. Press **Load**.
6. Navigate to and open your default private key.
7. Enter your passphrase when prompted and press OK.
   The system displays your public key.

8. Select and copy the contents of the **Public key for pasting into OpenSSH authorized_keys file** field.
9. Back in your browser, paste the contents into the field provided and press **Add key**.

    The system adds the key to your account.
10. Close PuTTYgen.

Using the Choose key button

If your browser has the **Choose key** button, you can do this instead:

1. In the **SSH keys** section, click **Choose key**.
   The system displays the **File Upload** dialog.
2. Navigate to your `C:\Users\username\.ssh` directory.
   This is a hidden directory. If hidden directories are invisible, set your file folder preferences to view hidden files and try again.
3. Locate the public key file (`filename.pub`) and click **Open**.
   The **Choose key** button changes to **Upload key** and the file name appears next to it.

4. Press **Upload key**.

Step 6. Configure your local repo to use the SSH protocol

The URL you use for a repo depends on which protocol you are using, HTTPS and SSH. The bitbucket repository **Overview** page has a quick way for you to see the one for your `myquotefork` repo. On the repo's **Overview** page look for the **Clone this repository** line. Experiment for a moment, click back and forth between the **SSH** and the **HTTPS** protocol links to see how the URLs differ. The table below shows the format for each DVCS based on protocol.

	SSH format	HTTPS format

Mercurial	`ssh://hg@bitbucket.org/accountname/reponame/`	`https://accountname@bitbucket.org/accountname/reponame`

Git	`git@bitbucket.org:accountname/reponame.git`
	
	or
	
	`ssh://git@bitbucket.org/accountname/reponame.git`
		`https://accountname@bitbucket.org/accountname/reponame.git`

Got to your local system and navigate to your `myquotefork` repo (the only Mercurial repo you've worked with so far). These instructions assume you have added the repo to the TortoiseHG Workbench.

1. Start TortoiseHG.
2. Right click your `myquotefork` repo and choose **Settings**.
   The system displays the **TortoiseHG Settings** dialog with the **myquotefork repository settings** tab active.
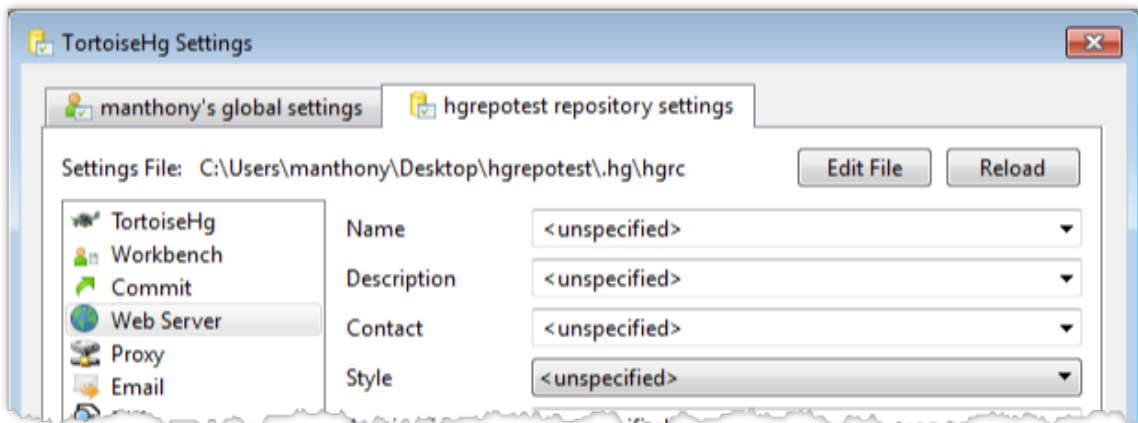
3. Press **Edit File**.
4. View your current repo configuration.
   You should see something similar to the following:

```
[paths]
default = https://bitbucket.org/newuserme/myquotefork
```

5. Change the `default` value to use the SSH format for that repo.
   When you are done you should see something similar to the following:

```
6. [paths]
   default = ssh://hg@bitbucket.org/newuserme/myquotefork
```

7. Press **Save** to close the editor.

8. Press **OK** to close the settings dialog.
9. Restart TortoiseHG Workbench so that it uses the new SSH setting.

Step 7. Make a change under the new protocol

1. Edit the `Index.html` file in your `myquotefork` repo.
2. Add a new line to the file.

3. Save and close the file.
4. Add and then commit your change to your local repo.
   A successful push shows in your TortoriseHG log as follows:

```
pushing to ssh://hg@bitbucket.org/newuserme/myquotefork
searching for changes
remote: adding changesets
remote: adding manifests
remote: adding file changes
remote: added 1 changesets with 1 changes to 1 files
remote: bb/acl: newuserme is allowed. accepted payload.
[command completed successfully Mon Dec 19 10:49:06 2011]
myquotefork%
```

5. Push your changes to your fork.
   PuTTY warns you that the host key is not yet stored.

6. Press **Yes** to add the bitbucket.org host key.
7. Open the repo **Overview** in bitbucket to view your commit.

Next Steps

You've completed the 101 tutorial for bitbucket. At this point, you should have a good beginners knowledge of what you can do in bitbucket. (You should also make sure you have checked https://tutorials.bitbucket.org to see your pull request changes incorporated.) If you are a Mac user, you might want to see Mac Users: SourceTree a Free Git and Mercurial GUI. The rest of the documentation has more topics and information that can help you make the most of bitbucket. Please let us know what you thought of this tutorial by logging an issue or by sending an email. Of course, you can always contribute by commenting on or editing a page directly.

Mac Users: SourceTree a Free Git and Mercurial GUI

SourceTree is a Mac GUI client you can use to quickly and easily view the status of Git, Mercurial, and Subversion source repositories. Atlassian recently acquired SourceTree which also provides special support for hosted DVCS systems such as bitbucket and GitHub. Using a single SourceTree client, you can work with local repositories and hosted repositories in multiple source control systems. If you are a Mac User, why not install SourceTree and give it a try to see if it is a fit for you?

Adding a hosted project in SourceTree

If you haven't already, start SourceTree. Then, do the following:

1. Click **Hosted Projects** from the menu bar.
   The system displays the **Hosted Repositories** dialog.
2. Press **Edit Accounts**.
   The system displays a list of the hosted accounts it knows about. If this is the first time you have run this, the list is empty.
3. Press **Add Account**.
4. Complete the dialog:

   The **Username** is your account name. The password you provide is your account password. You can choose whether you prefer connecting through HTTPS or SSH.
5. Press **OK**.
   SourceTree returns you to the list of hosted accounts. You can add another if you like.
6. Press **Close** when you are done adding accounts.
   The system displays a list of your hosted repositories.
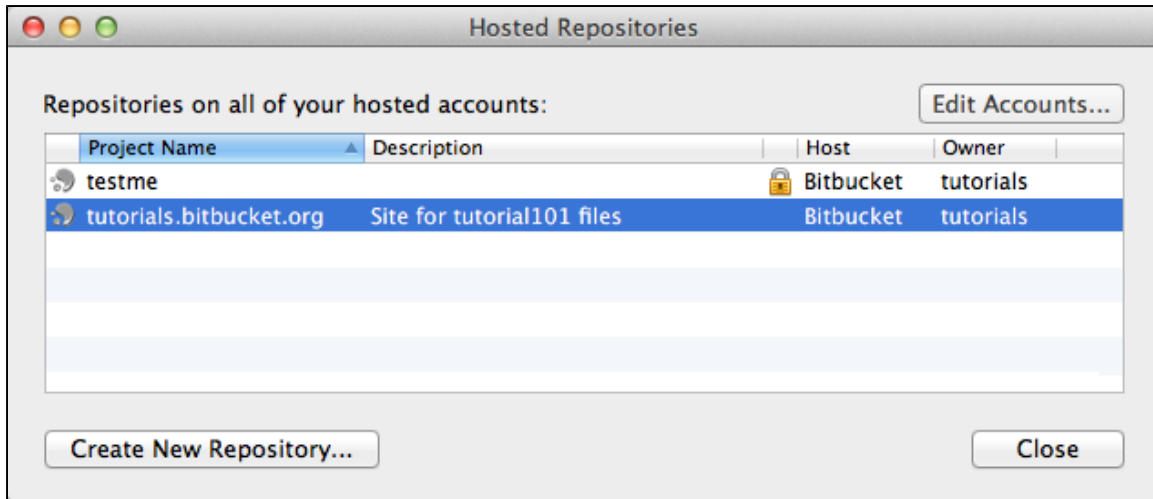
7. Press **Close**.

You'll notice that the SourceTree does not list your hosted repositories on your **Bookmarks** list. The bookmarks list is only for local repositories; you'll need to clone from a hosted repository or create a local repository to add to this list.

Clone a hosted project

To clone a hosted project to your local machine, you can do the following:

1. Press **Hosted Projects**.
   The system displays your hosted projects.
2. Select a project on the list.
3. Right-click to show the context menu.
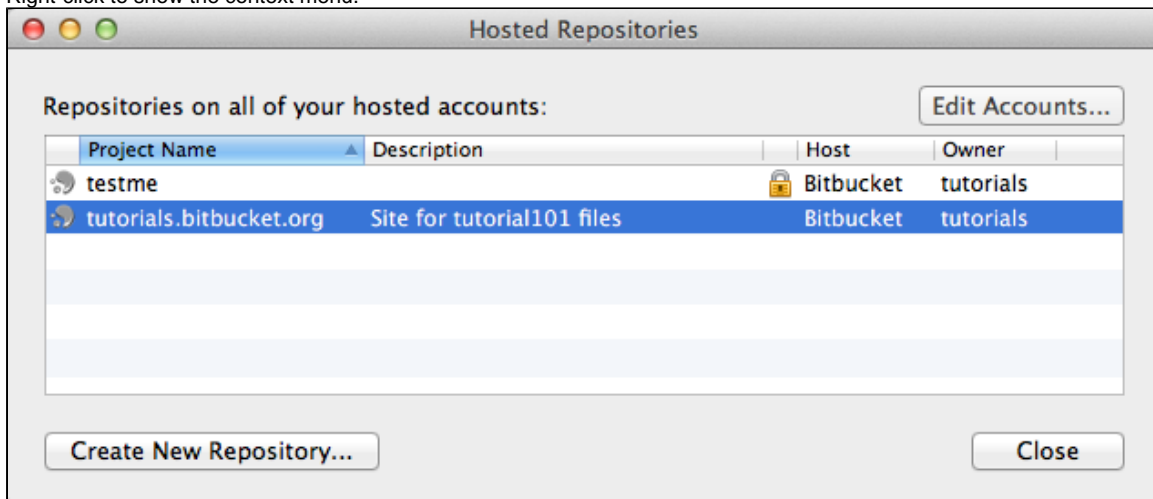
4. Click **Clone**.
   The system adds the clone to your list of bookmarks.

Doing more with SourceTree

There is a lot more you can do with SourceTree as you can see with this repository viewer:
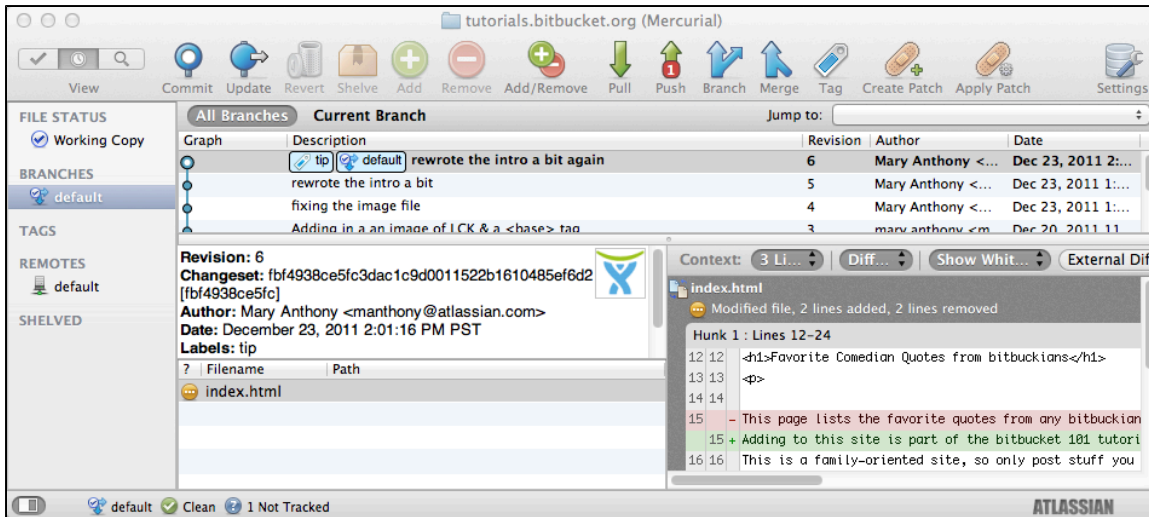
To learn more, try SourceTree yourself or review the Help that comes with it.

Importing code from an existing project

You can import code from an existing project into bitbucket. You can do this using the bitbucket importer or you can do this by pushing code from your local system to an empty bitbucket repository.

> ℹ️ If you are interested in migrating away from Subversion to Mercurial and bringing your history along, see Goodbye Subversion, Hello Mercurial: A Migration Guide in Atlassian Blogs.

Using the bitbucket importer to import from a hosting site or project

You can import from popular code hosting sites like GitHub or SourceForge, or from Git, Google Code, Mercurial or Subversion repositories, via the bitbucket importer. The importer cannot convert Git repositories to Mercurial repositories or vice versa. When you import code from an external Git or Mercurial repository, the importer simply creates a clone of the repository which is then hosted in bitbucket.

bitbucket does not host Subversion repositories. When you import from a Subversion repository, the importer:

* exports a working copy of your trunk code base
* builds a new DVCS repository
* commit the entire working copy as a single DVCS changeset into the new repository on bitbucket.

bitbucket does not the history when importing code from Subversion. If you would like to retain your Subversion revision history in bitbucket, you must perform an offline synchronisation. This may take a long time to complete and be CPU intensive, depending on the size of your repository, the number of revisions, branches and tags in your repository.

To import code do this.

1. Select the **Source** of the code you want to import.
2. Depending on the **Source**, the system asks you to provide the following information:

Source	Information you must supply
CodePlex	**URL**, **Project name**, **Repository type**
Git/GitHub	**URL**, a **Username/Password** combination for private repositories that **Require authorization**
Google Code	**URL**, **Project name**, **Repository type**
Mercurial	**URL**, a **Username/Password** combination for private repositories that **Require authorization**
SourceForge	**URL**, **Project name**, **Repository type**
Subversion	**URL**, a **Username/Password** combination for private repositories that **Require authorization**

3. Enter a **Name** for your new repository.
4. Uncheck **Private** if you want the repository to be **Public**.
5. Select a **Language**.
6. Click **Issue tracking** and/or **Wiki** if you want those features.
7. Enter a **Description** and **Website** if you wish.
8. Press **Import**.

Uploading or pushing a project to an empty bitbucket repository

You can upload an existing repo to a empty project in bitbucket. When you do this, bitbucket maintains your commit history.

Pushing a Git project

⊖ This kind of push overwrites the contents of the bitbucket repository. You should use it with great caution.

Before you can run this procedure, you must already added your SSH public key to your bitbucket account.

1. Create an empty repository in bitbucket.
2. Open a shell on your local machine (GitBash terminal for Windows users).
3. Verify your SSH key is working.

```
$ ssh -T git@bitbucket.org
conq: logged in as tutorials.

You can use git or hg to connect to Bitbucket. Shell access is disabled.
```

The message should report you are logged in as your bitbucket account. In this example, my ssh key was on my `tutorials` account. If you don't get this message, stop and troubleshoot your SSH connection to bitbucket. (See Using the SSH protocol with bitbucket for information about doing this.)

4. Navigate to the root directory of the repository you want to push.

```
$ cd ~/repos/originalrepo
```

5. Push the local repo up to bitbucket

```
$ git push --mirror git@bitbucket.org:tutorials/exploratory
```

This command pushes your local repo to the bitbucket server.

Pushing a Mercurial project

⊖ This kind of push overwrites the contents of the bitbucket repository. You should use it with great caution.

1. Create a new, empty repository on bitbucket.
2. Go to your local machine and start the TortoiseHg Workbench.
3. Select the repository you want to push from the **Repository Registry**.
4. Choose **File > Settings** from the menu bar.
   The TortoiseHg Settings dialog appears. It includes a tab for your global settings and a tab for the repository you selected.
5. Select the tab for the repository settings.
6. Click **Edit File**.
7. Change the `default` value.
   This value is the repo on bitbucket you want to push to:
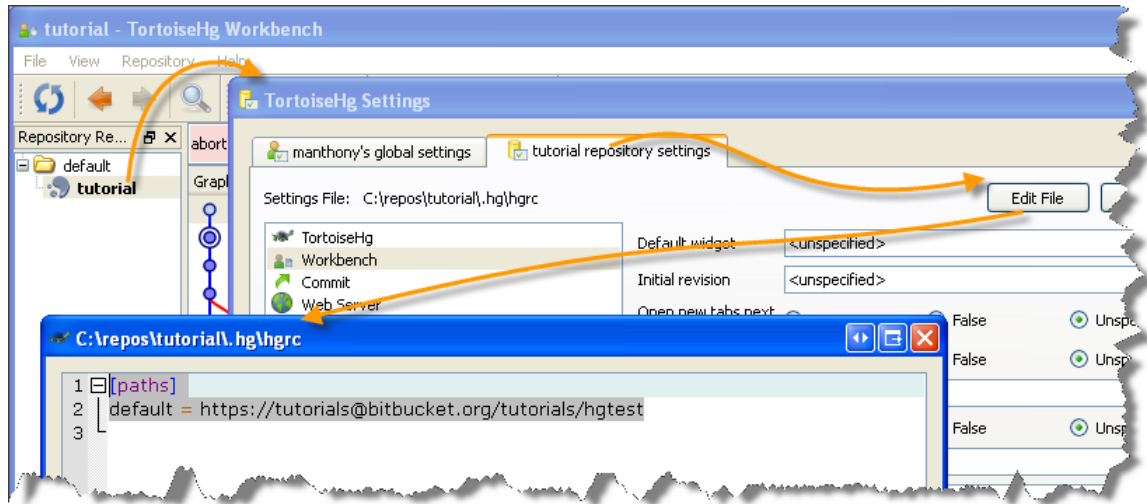
8. Press **Save** to save your edits and close the file.
9. Press **OK** to close the settings.
10. Press **Push outgoing changes to selected UR**L.

No TortoiseHg Workbench Available?

The TortoiseHg Workbench was introduced in version 2.0. If you don't have this version of TortoiseHg, you can upgrade. Alternatively, you can:

1. Change to the root of your Mercurial repository.
2. Change to the `.hg` subdirectory.
3. Edit the `hgrc` file.
4. Edit the default value making sure to specify the address of the empty repository you created in Step 1.
5. Save and close the file.
6. Push the repository.

Credits

The following users contributed tutorials that provided information which improved this page. Thank you.

- alombarte who wrote a tutorial which provided information for me to write this.
- Eli Spizzichino who also wrote a tutorial.

After I redid this page, I deleted the old comments that no longer applied. You can find them here as an attachment.

Using the SSH protocol with bitbucket

You can use either secure hypertext transport protocol (HTTPS) or secure shell (SSH) to connect to bitbucket. HTTPS requires you to enter a username/password each time you connect to the bitbucket server, for example, when you push your changes. HTTPS is suitable for situations where you work with bitbucket infrequently and make few code changes. However, if you do most of your coding with a bitbucket hosted repo, you'll want to set up a SSH connection. After you configure SSH, bitbucket no longer requires you to authenticate each remote communication with a username/password combination.

How SSH and bitbucket work together

To use SSH with bitbucket, you create an SSH identity. An identity consists of a private and a public key which together are a key pair. The private key resides on your local computer and the public you upload to your bitbucket account. Once you upload a public key to your account, you can use SSH to connect with repos you own and repos owned by others, provided those other owners give your account permissions. By setting up SSH between your local system and the bitbucket server, your system uses the key pair to automate authentication; you won't need to enter your password each time you interact with your bitbucket repo.

There are a few important concepts you need when working with SSH identities and bitbucket

- You cannot reuse an identity's public key across accounts. If you have multiple bitbucket accounts, you must create multiple identities and upload their corresponding public keys to each individual account.
- You *can* associate multiple identities with a bitbucket account. You would create multiple identities for the same account if, for example, you access a repo from a work computer and a home computer.   You might create multiple identities if you wanted to execute DVCS actions on a repo with a script – the script would use a public key with an empty passphrase allowing it to run without human intervention.
- RSA (R. **R**ivest, A. **S**hamir, L. **A**dleman are the originators) and digital signature algorithm (DSA) are key encryption algorithms. bitbucket supports both types of algorithms. You should create identities using whichever encryption method is most comfortable and available to you.

If you have multiple identities, you'll need a SSH *authentication agent* program on your local system. This program runs in the background. You load all your keys into the agent and it manages the authentication for you. In a Windows environment, a common authentication agent

is Pageant. In Mac OSX and Linux systems, `ssh-agent` is more common.

Repository URL formats by connection protocol

The URL you use to access a repository depends on the connection protocol (HTTPS or SSH) and the DVCS program. The following table shows the URL format for each case bitbucket supports:

	SSH format	HTTPS format
Mercurial	`ssh://hg@bitbucket.org/`*accountname*`/`*reponame*`/`	`https://`*accountname*`@bitbucket.org/`*accountname*`/`*reponame*

Git	`git@bitbucket.org:`*accountname*`/`*reponame*`.git`
	
	or
	
	`ssh://git@bitbucket.org/`*accountname*`/`*reponame*`.git`
		`https://`*accountname*`@bitbucket.org/`*accountname*`/`*reponame*`.git`

bitbucket displays repository-specific URLs for both Git and Mercurial on your repository **Overview** page.

Known host or bitbucket's public key fingerprints

Each server that allows connection over the SSH provides its own public key to the connecting client. SSH stores the key in a known hosts list. In Mac OSX and Linux this list is in a `~/.ssh/known_hosts` file. On Windows, PuTTYgen stores the information in the `HKEY_CURRENT_USER\SoftWare\SimonTatham\PuTTY\SshHostKeys` registry key. Each time your client reconnects SSH checks the server's identity against your known hosts just as the host is checking *your* identity. This two-way mechanism prevents man-in-the-middle attacks.

bitbucket hosts only allow Git and Mercurial to make SSH connections. The first time you access bitbucket using the SSH URL, your SSH client checks to see if the bitbucket host is a *known host*. If the host is not in your `~/.ssh/known_hosts` file SSH warns you that it is adding the bitbucket host to known hosts:

```
$ hg clone ssh://hg@bitbucket.org/newuserme/mquotefork testkey
The authenticity of host 'bitbucket.org (207.223.240.182)' can't be established.
RSA key fingerprint is 97:8c:1b:f2:6f:14:6b:5c:3b:ec:aa:46:46:74:7c:40.
Are you sure you want to continue connecting (yes/no)?
```

If you view the contents of known hosts is stored you find the actual key is stored in a base64 encoded format:

```
bitbucket.org,207.223.240.182 ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAubiN81eDcafrgMeLzaFPsw2kNvEcqTKl/VqLat/MaB33pZy0y3rJZtnqwR2qOOvbwKZYKiE(
```

Technically, you should record the server's public host key before connecting to it for the first time. Depending on the security protocols in your network, the system administrator may maintain a centrally located list of approved known hosts. The public key fingerprints for the bitbucket server are:

```
97:8c:1b:f2:6f:14:6b:5c:3b:ec:aa:46:46:74:7c:40 (RSA)
35:ee:d7:b8:ef:d7:79:e2:c6:43:9e:ab:40:6f:50:74 (DSA)
```

To get the format suitable for storage in the known hosts, you can use the following `ssh-keygen` command:

```
$ ssh-keyscan -t rsa bitbucket.org
# bitbucket.org SSH-2.0-OpenSSH_5.3
bitbucket.org ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAubiN81eDcafrgMeLzaFPsw2kNvEcqTKl/VqLat/MaB33pZy0y3rJZtnqwR2qOOvbwKZYKiE(
```

Basic setup with a single default identity

The bitbucket 101 shows you how to setup and use a single default identity with bitbucket. If you have completed the 101, much of what you

have read so far may seem familiar. If you have not completed the 101, you should at least work through the sections on SSH. These sections are:

- Set up SSH for Git (Windows/GitBash)
- Set up SSH for Mercurial (Windows/TortoiseHg)
- Set up SSH for Git and Mercurial (Mac OSX/Linux)

The section on Windows with TortoiseHG assumes you have installed PuTTy.

SSH multiple identities and other topics

Read the following topics for more information about using SSH with bitbucket:

- Configuring Multiple SSH Identities for GitBash, Mac OSX, & Linux
- Configuring Multiple SSH Identities for TortoiseHg
- Troubleshooting SSH Issues

Configuring Multiple SSH Identities for GitBash, Mac OSX, & Linux

Typically, if you are working with multiple accounts and/or multiple machines, you benefit from creating multiple SSH identities. In Mac OSX, GitBash, and Linux you can use the three ssh- commands to create and manage your identities.

SSH Command	Purpose
`ssh-keygen`	Creates key pairs.
`ssh-agent`	Agent for providing keys to remote servers. The agent holds loaded keys in memory.
`ssh-add`	Loads a private key into the agent.

To support multiple SSH identities in Bitbucket, do the following:

- Create multiple identities for Mac OSX, GitBash, and Linux
- Create a SSH config file
- Configure compression for Mercurial
- Load each key into the appropriate Bitbucket account
- Ensure the ssh-agent is running
- Clone a repository using SSH
- Change existing repositories to from HTTPS to SSH (optional)

Acknowledgement
My thanks to the codingbadger and to Charles on the bitbucket team who helped me get my head around this technique. Any errors here are of my own making of course.

Create multiple identities for Mac OSX, GitBash, and Linux

You should at this point already have created at least a single default identity. To see if you have a default identity already, list the contents of your .ssh directory. Default identity files appear as a `id_encrypt` and `id_encrypt.pub` pair. The `encrypt` value is either `rsa` or `dsa`. Use the `ssh-keygen` command to create a new identity. In the example below, the identity is named `personalid`.

```
    $ ssh-keygen -f ~/.ssh/personalid -C "personalid"
    Generating public/private rsa key pair.
    Enter passphrase (empty for no passphrase):
    Enter same passphrase again:
    Your identification has been saved in /Users/manthony/.ssh/personalid.
    Your public key has been saved in /Users/manthony/.ssh/personalid.pub.
    The key fingerprint is:
    7a:9c:b2:9c:8e:4e:f4:af:de:70:77:b9:52:fd:44:97 personalid
    The key's randomart image is:
    +--[ RSA 2048]----+
    |                 |
    |                 |
    |               .|
    |              Eo|
    |    .   S   . ..|
    |  . . o .  ... .|
    |   . = = ..o  o |
    |  . o X ... .  .|
    |   .ooB.o  ..   |
    +-----------------+
```

If you have multiple Bitbucket accounts, you need to generate a new public/private key pair for each account.

Create a SSH config file

When you have multiple identity files, create a SSH `config` file mechanisms to create aliases for your various identities. You can construct a SSH `config` file using many parameters and different approaches. The format for the alias entries use in this example is:

```
Host alias
  HostName bitbucket.org
  IdentityFile ~/.ssh/identity
```

To create a config file for two identities (workid and personalid), you would do the following:

1. Open a terminal window.
2. Edit the `~/.ssh/config` file.
   If you don't have a `config` file, create one.
3. Add an alias for each identity combination for example:

   ```
      Host workdid
       HostName bitbucket.org
       IdentityFile ~/.ssh/workdid
      Host personalid
       HostName bitbucket.org
       IdentityFile ~/.ssh/personalid
   ```

4. Close and save the file.

Now, you can substitute the alias for portions of the repository URL address as illustrated in the following table:

DVCS	Default address	Address with alias
Git	`git@bitbucket.org:`*accountname*`/`*reponame*`.git`	`git@`**alias**`:`*accountname*`/`*reponame*`.git`
Mercurial	`ssh://hg@alias/`*username*`/`*reponame*`/`	`ssh://hg@`**alias**`/`*username*`/`*reponame*`/`

There are lots of ways to use SSH aliasing. Another common use case may be the situation where you are using bitbucket and GitHub on the same machine. The codingbadger suggested the following configuration for that use case:

```
# Default GitHub user
Host github.com
  HostName github.com
  PreferredAuthentications publickey
  IdentityFile ~/.ssh/id_rsa

# Work user account
Host bitbucket.org
  HostName bitbucket.org
  PreferredAuthentications publickey
  IdentityFile ~/.ssh/id_rsa_work
```

If you google for "ssh aliases" or "ssh aliasing" you may find examples that suit you needs better.

Configure compression for Mercurial

When sending or retrieving data using SSH, Git does compression for you. Mercurial does not automatically do compression. You should enable SSH compression as it can speed up things drastically, *in some cases*. To enable compression for Mercurial, do the following:

1. Open a terminal window.
2. Edit the Mercurial global configuration file (`~/.hgrc`).
3. Add the following line to the UI section:

```
ssh = ssh -C
```

When you are done the file should look similar to the following:

```
[ui]
# Name data to appear in commits
username = Mary Anthony <manthony@atlassian.com>
ssh = ssh -C
```

4. Save and close the file.

Load each key into the appropriate Bitbucket account

You load each identities public key into corresponding account. If you have multiple Bitbucket accounts, you load each account with the corresponding public key you created. If you have an account with a repository you access from two identities, you can load two keys into that account – one for each identity. Use the following procedure to load each key into your Bitbucket accounts:

1. Open a browser and log into bitbucket.
2. Choose *Username* > **Account** from the menu bar.
   The system displays the **Account settings** page.
3. Locate the **SSH keys** section and the **Add key** field.
4. In your terminal window, `cat` the contents of the public key file.
   For example:

```
cat ~/.ssh/id_rsa.pub
```

5. Select and copy the output.

   In Mac OSX the following command copies `cat` output to the clipboard:

```
cat ~/.ssh/id_rsa.pub | pbcopy
```

6. Paste the captured key into the field provided and press **Add key**.

The system adds the key to your account.

Using the Choose key button

If your browser has the **Choose key** button, you can do this instead:

1. In the **SSH keys** section, click **Choose key**.
   The system displays the **File Upload** dialog.
2. Navigate to your `/Users/username/.ssh` directory.
   This is a hidden directory. If hidden directories are invisible, in Mac OSX press **command+shift+.** (period) . In Windows set your file folder preferences to view hidden files and try again.
3. Locate the public key file (`filename`.pub) and click **Open**.
   The **Choose key** button changes to **Upload key** and the file name appears next to it.

4. Press **Upload key**.

Ensure the ssh-agent is running

Most modern operating systems (and GitBash) start a ssh-agent running for you. However, it is important you know how to check for a running agent and start one if necessary. Open a terminal window and enter the appropriate command for your operating system.

GitBash	Mac OSX andLinux
``$ ps	grep ssh-agent``` `5192   1    5192      5192    ?  500 19:23:34 /bin/ssh-agent` If for some reason the agent isn't running, start it by entering `ssh-agent` at the command line. You should only be running a single instance of ssh-agent. If you have multiple instances running, use the `kill PID` command to stop each of them. Then, restart a single instance.	```$ ps -e	grep [s]sh-agent``` `9060 ?? 0:00.28 /usr/bin/ssh-agent -l` If the agent isn't running, start it by hand. The format for starting the command manually is: `$ ssh-agent $SHELL` `$SHELL` is the environment variable for your login shell.

Clone a repository using SSH

To clone a repository with one of multiple SSH identities that you have added to an SSH `config`, you would log into Bitbucket and do the following:

1. Navigate to the repository **Overview**.
2. Display the SSH URL.
   For example, Bitbucket displays its tutorial URL as:

   `hg clone ssh://hg@bitbucket.org/tutorials/tutorials.bitbucket.org`

3. Open a terminal window on your system.

4. Navigate to the directory where you store your repos.

5. Enter the command but substitute your `config` alias appropriately:

   ```
   hg clone ssh://hg@personalid/tutorials/tutorials.bitbucket.org
   ```

   The system clones the repo for you.

6. Change directory to the repo.

7. Display the contents of the repo's configuration.

```
$ cat .hg/hgrc
[paths]
default = ssh://hg@personalid/tutorials/tutorials.bitbucket.or
```

Notice that the DVCS stored the URL you used for the clone. Now, moving forward for this repo, the DVCS uses the URL that includes the SSH alias.

Change existing repositories to from HTTPS to SSH (optional)

You can change existing repo configurations to use a SSH configuration that makes use of your multiple identities. You'll only need to do this for repos that you have already cloned with HTTPS or for repos where you want to change an existing SSH specification. For example, if you used SSH to clone a repository in the past and now want to set it up to use another SSH key.

Git configuration

1. Open a terminal window.
2. Navigate to the repo configuration file (REPO_INSTALLDIR/.git).

3. Open the config file with your favorite editor.

4. Locate the url value in the [remote "origin"] section

```
[remote "origin"]
    fetch = +refs/heads/*:refs/remotes/origin/*
    url = https://newuserme@bitbucket.org/newuserme/bb101repo.git
```

In this example, the url is using the HTTPS protocol.

5. Change the url value to use the SSH format for your repo.
   When you are done you should see something similar to the following:

```
[remote "origin"]
    fetch = +refs/heads/*:refs/remotes/origin/*
    url = git@personalid:newuserme/bb101repo.git
```

Mercurial Configuration

1. Open a terminal window.
2. Navigate to the repo configuration file (REPO_INSTALLDIR/.hg).

3. Open the hgrc file with your favorite editor.

```
[paths]
default = https://newuserme@staging.bitbucket.org/newuserme/bb101repo
```

4. Change the [paths] default to:

```
[paths]
default = ssh://hg@personalid/newuserme/bb101repo
```

5. Save and close the file.

Configuring Multiple SSH Identities for TortoiseHg

If you are using TortoiseHg on Windows, you need the following programs to use SSH:

Program	Description
PuTTYgen	An RSA and DSA key generation utility. This is a part of the free PuTTY telnet/SSH Client. You can install this package separately from TortoiseHg. You can also install just PuTTYgen.exe into the same directory as TortoiseHg.

| Pageant | An SSH authentication agent for PuTTY, PSCP, PSFTP, and Plink. This is installed for you with TortoiseHg. |

TortoiseHG connects with Plink and uses Pageant to identify which keys are available. To configure TortoiseHg so that you can access multiple bitbucket accounts through SSH, do the following:

- Generate a key pair for a new identity
- Add your private keys to Pageant
- Add your SSH public key to bitbucket
- Configure Mercurial Compression
- Clone a repo as you normally would
- Switching existing repos from HTTPS to SSH (optional)

If you have a single identity and that is all you need, you can configure a single, global identity instead.

Generate a key pair for a new identity

PuTTYgen creates an identity for both public and private use. You save the private key in a **.ppk** file. You can also save the public key to a file but that isn't strictly necessary. These instructions assume you are only saving the private key file.

1. Locate the `puttygen.exe` executaible in your system and double click the icon to start it.
   The system opens the **PuTTY Key Generator** dialog.
2. Press **Generate**.
   Following the instructions to generate some randomness. When the generation completes, the system displays the public key and a number of other fields:

3. Edit the **Key comment**.
   The comment appears in the key list on bitbucket. Using this comment is a great way to distinguish your keys.
4. Enter and confirm a key passphrase.

5. Press **Save private key**.
   The system prompts you for a location to save the file and a file name. By convention, store your key files in a folder called `C:\Users\`*`yourname`*`\.ssh`. and give it a `.pub` extension.

6. Close the key generator.

Add your private keys to Pageant

1. Start Pageant by double clicking its icon.
   By default, TortoiseHG installs the Pageant in the `C:\Program Files\TortoiseHG` folder. When it is running, Pageant appears in your system tray:
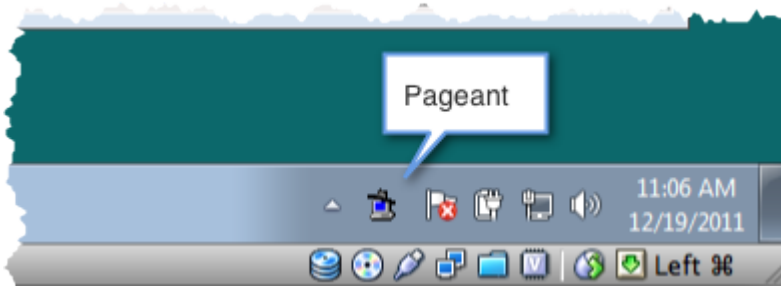
2. Double-click the Pageant icon to launch the **Pagent Key List** dialog.
3. Click the **Add Key** button.
   The system displays the **Select Private Key File** dialog.
4. Navigate to and open the public key file.
5. Enter the passphrase when prompted:
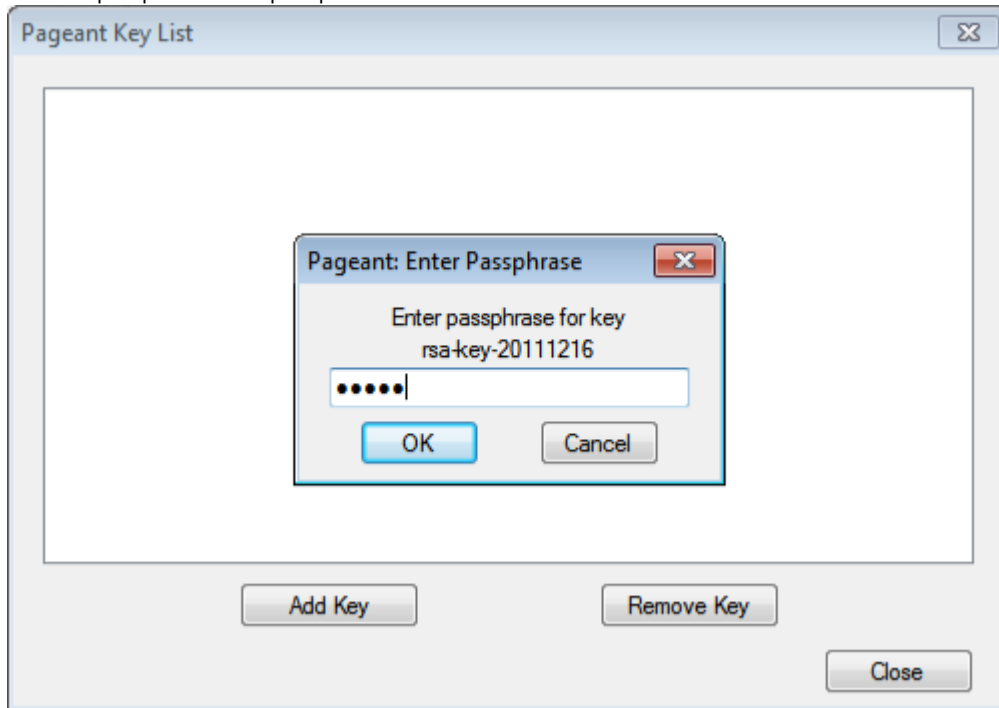
6. Press **OK**.
   Pageant shows your key in the running list.
7. Add any additional keys.
8. Press **Close** to close the dialog.
   Pageant continues to run on your system.

Add your SSH public key to bitbucket

1. Open a browser and log in to bitbucket.
2. Choose *Username* > **Account** from the menu bar.
   The system displays the **Account settings** page.
3. Locate the **SSH keys** section and its **Add key** field.
   You will need to paste your public key into the Add key field.
4. Start the PuTTYgen program.
5. Press **Load**.
6. Navigate to and open your default private key.
7. Enter your passphrase when prompted and press OK.
   The system displays your public key.

8. Select and copy the contents of the **Public key for pasting into OpenSSH authorized_keys file** field.
9. Back in your browser, paste the contents into the field provided and press **Add key**.

The system adds the key to your account.
10. Close PuTTYgen.

Using the Choose key button

If your browser has the **Choose key** button, you can do this instead:

1. In the **SSH keys** section, click **Choose key**.
   The system displays the **File Upload** dialog.
2. Navigate to your `C:\Users\username\.ssh` directory.
   This is a hidden directory. If hidden directories are invisible, set your file folder preferences to view hidden files and try again.
3. Locate the public key file (`filename.pub`) and click **Open**.
   The **Choose key** button changes to **Upload key** and the file name appears next to it.

4. Press **Upload key**.

Configure Mercurial Compression

When sending or retrieving data using SSH, Git does compression for you. Mercurial does not automatically do compression. If you are using Mercurial, you should enable SSH compression as it can speed up things drastically, *in some cases*. To enable compression for Mercurial, do the following:

1. Start the TortoiseHg Workbench.
2. Select **File > Settings**.
3. Make sure you have the global settings tab selected.
4. Press **Edit File**.
5. Add the following line to the UI section:

```
ssh = "TortoisePlink.exe" -ssh -2 -batch -C
```

When you are done the file should look similar to the following:

```
[ui]
# Name data to appear in commits
username = Mary Anthony <manthony@atlassian.com>
ssh = "C:\Program Files\TortoiseHg\TortoisePlink.exe" -ssh -2 -batch -C
```

6. Press **Save** to store your settings and close the file.
7. Press **OK** to close the settings dialog.

Clone a repo as you normally would

1. Start TortoiseHG Workbench.
2. Select **File > Clone Repository**.
3. Enter the proper SSH syntax (you can copy this from bitbucket's repo **Overview** page) in the **Source** field.
4. Enter a **Destination** for your repo.
   At this point the clone dialog looks similar to the following:
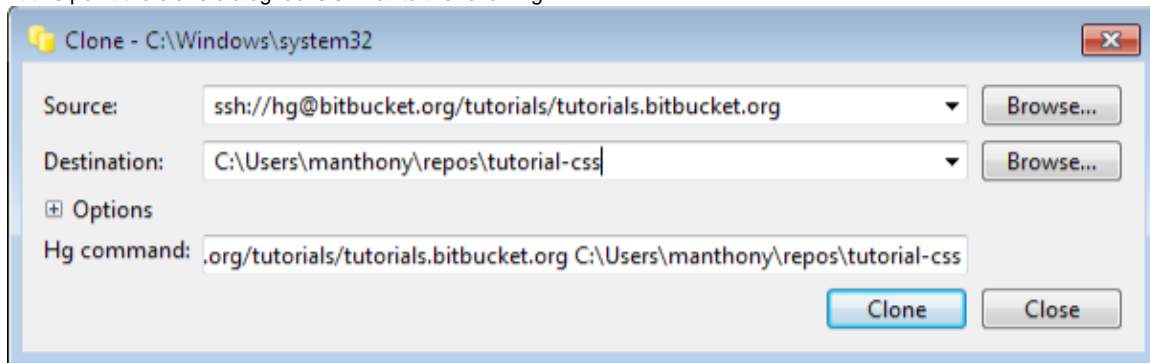
5. Press **Clone**.
   The system clones the repository and adds it to your repo registry in TortoiseHg.
6. Right click the new repo in the registry and choose **Settings**.
   The system displays the **TortoiseHG Settings** dialog with the *name* **repository settings** tab active.
7. Press **Edit File**.
8. View your current repo configuration.
   You should see something similar to the following:

```
[paths]
default = ssh://hg@bitbucket.org/newuserme/bb101repo
```

Go ahead and close the file and the settings dialog.

Switching existing repos from HTTPS to SSH (optional)

You can change existing repo configurations to use a SSH configuration that makes use SSH. You'll only need to do this for repos that you have already cloned with HTTPS.

1. Start TortoiseHG Workbench.
2. Right click a repo and choose **Settings**.
   The system displays the **TortoiseHG Settings** dialog with the *name* **repository settings** tab active.
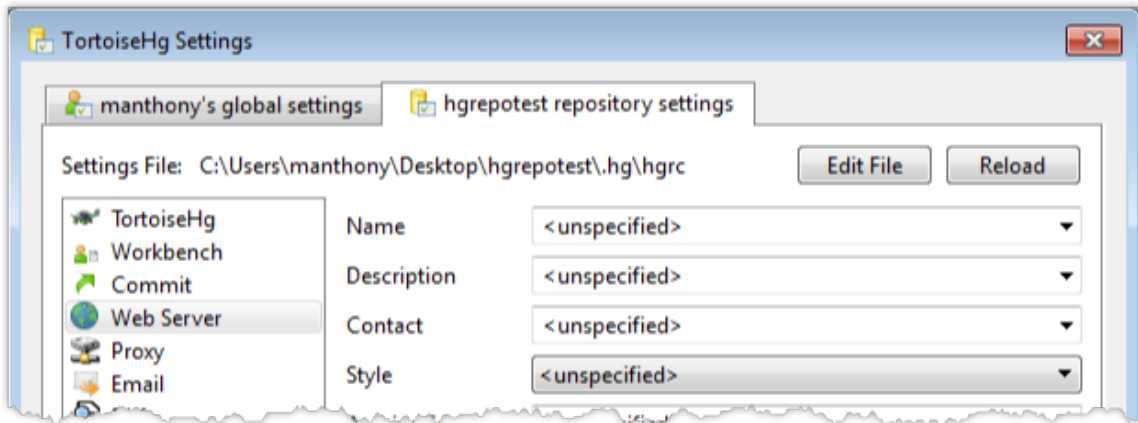
3. Press **Edit File**.
4. View your current repo configuration.
   You should see something similar to the following:

   ```
   [paths]
   default = https://bitbucket.org/accountname/reponame
   ```

5. Change the `default` value to use the SSH format for that repo.
   When you are done you should see something similar to the following:

   ```
   [paths]
   default = ssh://hg@bitbucket.org/accountame/reponame
   ```

6. Press **Save** to close the editor.

7. Press **OK** to close the settings dialog.

Configure a single, global identity

If you only plan on ever using a single identity with your TortoiseHG installation, you can create a global configuration for yourself. To do this, start the TortoiseHG Workbench and do the following:

1. Choose **File > Settings**.
   The system opens the settings dialog.
2. Select the *username* **global settings** tab.
3. Press **Edit File**.
   The system opens an editor with the `C:\Users\`*username*`\mercurial.ini` file.
4. Edit the [ui] section and add an identity to the TortoisePlink call:

   ```
   # Generated by TortoiseHg setting dialog
   [ui]
   username = yourusername <youremail@yourdomain.ext>
   ssh = "C:\Program Files (x86)\TortoiseHg\TortoisePlink.exe" -ssh -2 -i
   C:\Users\yourusername\.ssh\keyname.ppk
   ```

5. Press **Save** to close the editor.
6. Press **OK** to close the dialog.

Plink has a number of command line options that TortoisePlink accepts. For more information, see the PuTTYgen documentation.

Troubleshooting SSH Issues

If you are having problems with SSH, here are some things you can try when troubleshooting your issues.

- General SSH troubleshooting
  - Is the ssh-agent running?
  - Is the identity you want to use loaded?
  - ssh -T connection test (GitBash/Mac OSX/Linux)
  - Using a single key without a config file (Mac OSX and Linux)
  - ssh-agent on GitBash
  - Using ssh verbose mode to track down problems

- Mercurial (or TortoiseHg)
    - remote: No supposted authentication methods left to try!
    - pushing a new repository is slow or hangs

General SSH troubleshooting

Use this section for general SSH troubleshooting

Is the ssh-agent running?

On Windows make sure Pageant is running in your system tray:
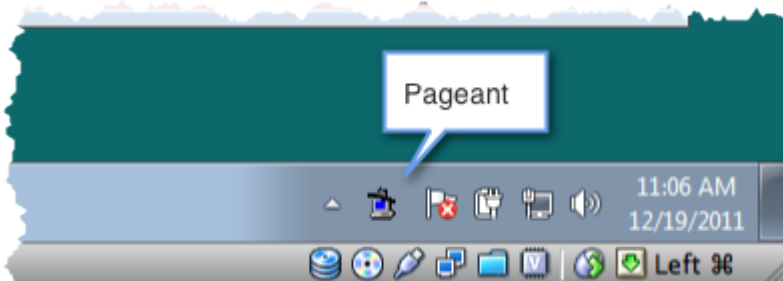
For GitBash, Mac OSX, or Linux:

Open a terminal window and enter the `ps -e | grep [s]sh-agent` command to see if the agent is running:

```
myhost:~ manthony$ ps -e  | grep [s]sh-agent
 9060 ??         0:00.28 /usr/bin/ssh-agent -l
```

If the agent isn't running, start it by hand. The format for starting the command manually is:

```
myhost:~ manthony$ ssh-agent $SHELL
```

`$SHELL` is the environment variable for your login shell.

Is the identity you want to use loaded?

On Windows, double-click Pageant and check which key is loaded:
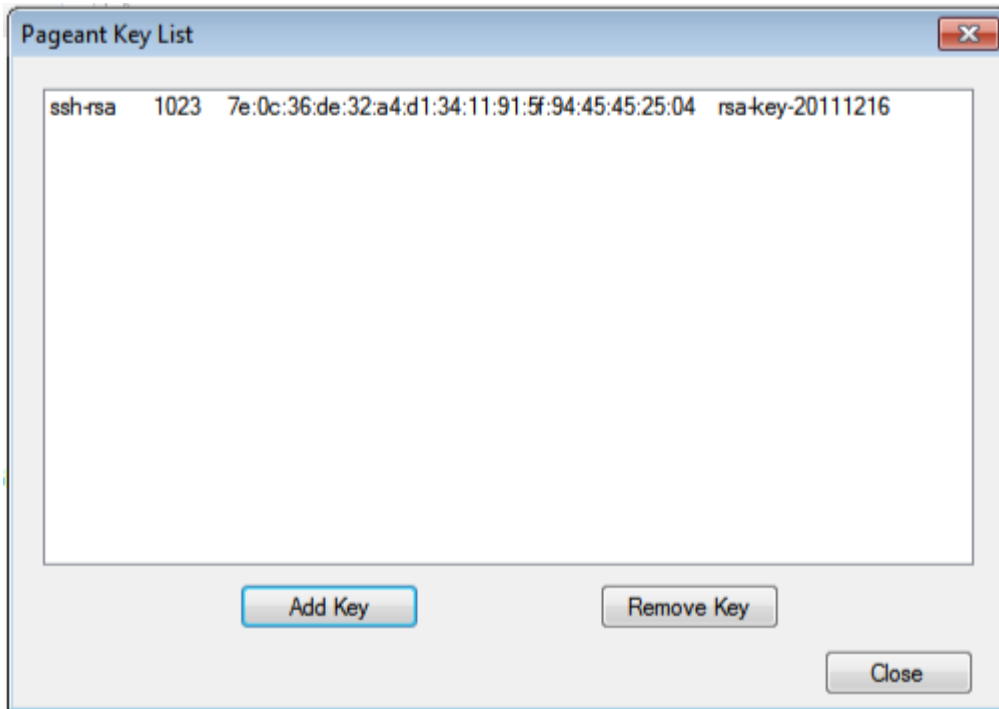
On GitBash/Mac OSX/Linux, the the following:

To list the loaded entities:

```
$ ssh-add -l
2048 4c:80:61:2c:00:3f:9d:dc:08:41:2e:c0:cf:b9:17:69 /Users/manthony/.ssh/workid (RSA)
2048 7a:9c:b2:9c:8e:4e:f4:af:de:70:77:b9:52:fd:44:97 /Users/manthony/.ssh/personalid (RSA)
```

To load an identity:

```
ssh-add ~/.ssh/identity
```

With `ssh-agent` management bitbucket uses the first key in the list. If you are still having problems, try removing all but the identity you want to connect with, to do this:

```
ssh-add -d ~/.ssh/identity
```

ssh -T connection test (GitBash/Mac OSX/Linux)

You can use the `ssh -T` command to test your connection to bitbucket. If you don't have any keys loaded in the agent, the test responds like this:

```
$ ssh -T hg@bitbucket.org
Permission denied (publickey).
```

A successful response looks like the following:

```
$ ssh -T hg@bitbucket.org
conq: logged in as tutorials.

You can use git or hg to connect to bitbucket. Shell access is disabled.
```

Of course, you can also test with a `ssh -T` git@bitbucket.org command structure.

Using a single key without a config file (Mac OSX and Linux)

Instead of the `config` file, you can use the `ssh-add` command to manually load an identity into the `ssh-agent` management program.

```
$ ssh-add ~/.ssh/personalid
Enter passphrase for /Users/manthony/.ssh/personalid:
Identity added: /Users/manthony/.ssh/personalid (/Users/manthony/.ssh/personalid)
myhost:~ manthony$
```

Use the `ssh-add` command to list the keys that the agent is managing:

```
$ ssh-add -l
2048 4c:80:61:2c:00:3f:9d:dc:08:41:2e:c0:cf:b9:17:69 /Users/manthony/.ssh/workid (RSA)
2048 7a:9c:b2:9c:8e:4e:f4:af:de:70:77:b9:52:fd:44:97 /Users/manthony/.ssh/personalid (RSA)
```

In this instance, the bitbucket server always uses the first key it encounters. So, this is not a good technique to use for multiple identities.

You can remove an identity from the agent using `ssh-add` with the `-d` flag. You can also set an identity to expire from the list with the `ssh-add` command.

ssh-agent on GitBash

If your key is uploaded and you are still having trouble connecting. Make sure you are not running multiple versions of the `ssh-agent`. To do this, enter the following at the GitBash command line:

```
$ ps
     PID    PPID    PGID    WINPID  TTY UID    STIME COMMAND
    5192       1    5192      5192    ?  500 19:23:34 /bin/ssh-agent
    5840       1    5840      5840  con  500 08:38:20 /bin/sh
    6116    5840    6116      1336  con  500 08:38:22 /bin/ps
```

Kill all the agents and restart it. Some users find that the ssh-agent binds to the terminal window they use effectively blocking any further action with SSH. To avoid this problem, run the program as follows:

```
$ eval 'ssh-agent'
```

This runs the process and also forks it from the terminal window.

Using ssh verbose mode to track down problems

You can use the `-v (verbose)` flag with ssh to track down problems with your connection. What happens if you receive a `Permission denied (publickey)` error but you verified your key is loaded both on your local system and into your bitbucket account? This happened to me recently.

Running the verbose command:

Output showing the error:

```
myhost:~ manthony$ ssh -v hg@bitbucket.org
OpenSSH_5.6p1, OpenSSL 0.9.8r 8 Feb 2011
debug1: Reading configuration data
/Users/manthony/.ssh/config
debug1: Applying options for bitbucket.org
debug1: Reading configuration data
/etc/ssh_config
debug1: Applying options for *
debug1: Connecting to bitbucket.org
[172.16.10.101] port 22.
debug1: Connection established.
debug1: identity file
/Users/manthony/.ssh/manthony type 1
debug1: identity file
/Users/manthony/.ssh/manthony-cert type -1
debug1: Remote protocol version 2.0, remote
software version OpenSSH_5.3
debug1: match: OpenSSH_5.3 pat OpenSSH*
debug1: Enabling compatibility mode for
protocol 2.0
debug1: Local version string
SSH-2.0-OpenSSH_5.6
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr
hmac-md5 zlib@openssh.com
debug1: kex: client->server aes128-ctr
hmac-md5 zlib@openssh.com
debug1:
SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<8192)
sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GROUP
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REPLY
debug1: Host 'bitbucket.org' is known and
matches the RSA host key.
debug1: Found key in
/Users/manthony/.ssh/known_hosts:2
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by server
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue:
publickey
debug1: Next authentication method:
publickey
debug1: Offering RSA public key:
/Users/manthony/.ssh/manthony
debug1: Authentications that can continue:
publickey
debug1: No more authentication methods to
try.
Permission denied (publickey).
```

In this run, the public key `manthony` was offered. It failed, the system then tried to use the default key and again failed. If the proper key is offered but fails, check and make sure the key is uploaded to your account. If the key is in the account but it is still failing, try uploading it again. You can also try another key.

If your key isn't offered, make sure the key exists on the local system you are using to connect. Make sure
your key is loaded into the key agent. You can do this via a `.ssh/config` file or manually loading it into
the SSH agent with `ssh-add`. To test if the key is loaded, with `ssh-add`
run the `ssh-add -l` command.
Finally, try recreating the key and uploading it again.

Output showing success:

```
myhost:~ manthony$ ssh -v hg@bitbucket.or
OpenSSH_5.6p1, OpenSSL 0.9.8r 8 Feb 2011
debug1: Reading configuration data
/Users/manthony/.ssh/config
debug1: Applying options for bitbucket.or
debug1: Reading configuration data
/etc/ssh_config
debug1: Applying options for *
debug1: Connecting to bitbucket.org
[172.16.10.101] port 22.
debug1: Connection established.
debug1: identity file
/Users/manthony/.ssh/manthony type 1
debug1: identity file
/Users/manthony/.ssh/manthony-cert type -
debug1: Remote protocol version 2.0, remo
software version OpenSSH_5.3
debug1: match: OpenSSH_5.3 pat OpenSSH*
debug1: Enabling compatibility mode for
protocol 2.0
debug1: Local version string
SSH-2.0-OpenSSH_5.6
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: server->client aes128-ctr
hmac-md5 zlib@openssh.com
debug1: kex: client->server aes128-ctr
hmac-md5 zlib@openssh.com
debug1:
SSH2_MSG_KEX_DH_GEX_REQUEST(1024<1024<819
sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_GRO
debug1: SSH2_MSG_KEX_DH_GEX_INIT sent
debug1: expecting SSH2_MSG_KEX_DH_GEX_REP
debug1: Host 'bitbucket.org' is known and
matches the RSA host key.
debug1: Found key in
/Users/manthony/.ssh/known_hosts:2
debug1: ssh_rsa_verify: signature correct
debug1: SSH2_MSG_NEWKEYS sent
debug1: expecting SSH2_MSG_NEWKEYS
debug1: SSH2_MSG_NEWKEYS received
debug1: Roaming not allowed by serverhg
debug1: SSH2_MSG_SERVICE_REQUEST sent
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue
publickey
debug1: Next authentication method:
publickey
debug1: Offering RSA public key:
/Users/manthony/.ssh/manthony
debug1: Remote: Forced command: conq
manthony
debug1: Remote: Port forwarding disabled.
debug1: Remote: X11 forwarding disabled.
debug1: Remote: Agent forwarding disabled
debug1: Remote: Pty allocation disabled.
debug1: Server accepts key: pkalg ssh-rsa
blen 279
debug1: PEM_read_PrivateKey failed
debug1: read PEM private key done: type
<unknown>
debug1: read PEM private key done: type R
Identity added:
/Users/manthony/.ssh/manthony
(/Users/manthony/.ssh/manthony)
debug1: read PEM private key done: type R
Connection closed by 172.16.10.101
```

Mercurial (or TortoiseHg)

Use this section if you are working in Microsoft windows with TortoiseHG and managing your keys with PuTTYgen/Pageant.

remote: No supposted authentication methods left to try!

You can get this error, for example, when cloning a repository:
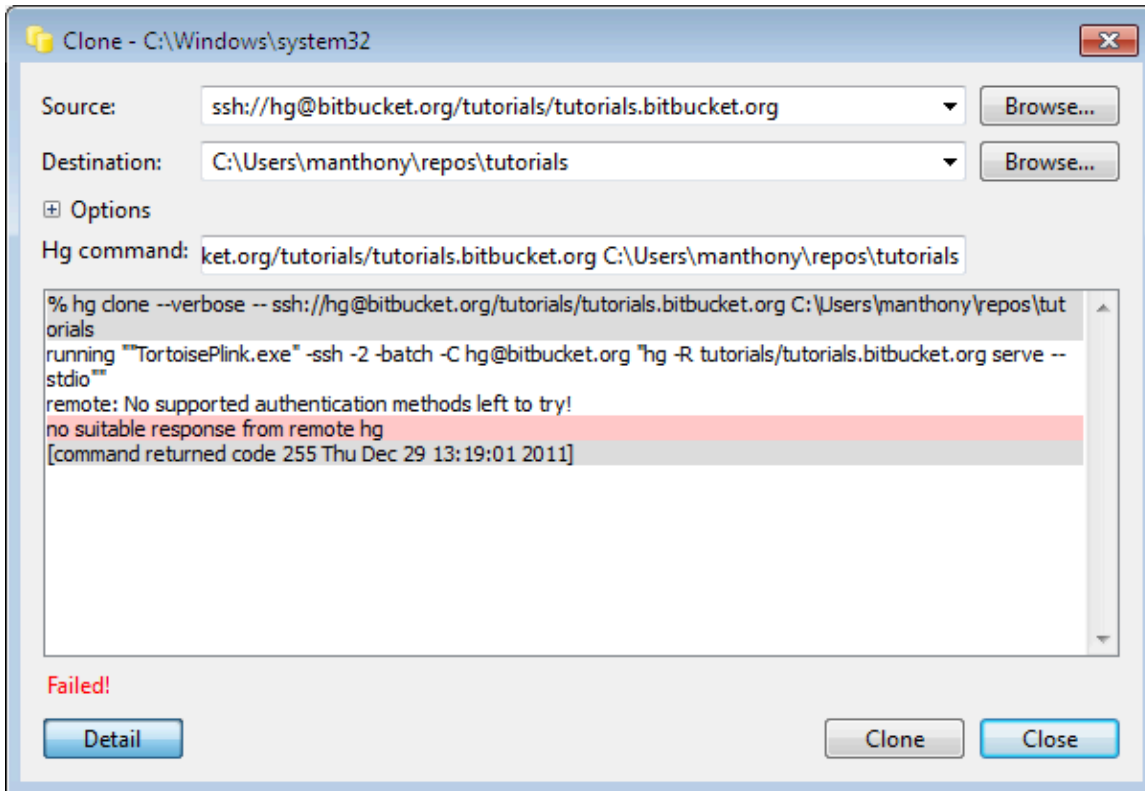
Check the following:

- Pageant is running and has a key loaded.
- You have added the corresponding public key into your bitbucket account.

pushing a new repository is slow or hangs

It could be you have not configured Mercurial compression. On Windows do this:

When sending or retrieving data using SSH, Git does compression for you. Mercurial does not automatically do compression. If you are using Mercurial, you should enable SSH compression as it can speed up things drastically, *in some cases*. To enable compression for Mercurial, do the following:

1. Start the TortoiseHg Workbench.
2. Select **File > Settings**.
3. Make sure you have the global settings tab selected.
4. Press **Edit File**.
5. Add the following line to the UI section:

```
ssh = "TortoisePlink.exe" -ssh -2 -batch -C
```

   When you are done the file should look similar to the following:

```
[ui]
# Name data to appear in commits
username = Mary Anthony <manthony@atlassian.com>
ssh = "C:\Program Files\TortoiseHg\TortoisePlink.exe" -ssh -2 -batch -C
```

6. Press **Save** to store your settings and close the file.
7. Press **OK** to close the settings dialog.

In Mac OSX or Linux do this:

When sending or retrieving data using SSH, Git does compression for you. Mercurial does not automatically do compression. You should enable SSH compression as it can speed up things drastically, *in some cases*. To enable compression for Mercurial, do the following:

1. Open a terminal window.
2. Edit the Mercurial global configuration file (`~/.hgrc`).
3. Add the following line to the UI section:

```
ssh = ssh -C
```

When you are done the file should look similar to the following:

```
[ui]
# Name data to appear in commits
username = Mary Anthony <manthony@atlassian.com>
ssh = ssh -C
```

4. Save and close the file.

Repository privacy, permissions, and more

The purpose of a hosted DVCS is, in part, to allow geographically dispersed users to collaborate on code development. To do this effectively in bitbucket, you need to understand public and private as it applies to repos. You also need to understand what types of permissions you can grant to users in your repos.

Public and Private Repositories

The user that creates a repo is known as its *owner*. Only an owner can delete a repo. You can create unlimited bitbucket public or private repositories. A public repository is visible to all bitbucket users. A private repository is visible only to its owner and users that owner gives permissions to.

Repository owners can grant other users explicit access to the repository. An owner can allow 5, 10, 25, 50, or an unlimited number of users access to repositories under his or her control. The owner always counts as one user. The owner configures the permissions associated with each user with access. bitbucket supports the following permission levels:

Level	This permissions allows a user to do the following:
read	View the repository contents. All public repositories grant all bitbucket users read permissions automatically.
write	Contribute to the repository by pushing changes directly from a repository on a local machine.
admin	Do everything a repository owner can do, except delete the repository. This means administrators can: • Change repository settings. • Add, change, and remove user permissions. • Give other users administrator access.

By default, all repositories public and private permit other users to create a public fork of that repository. If you haven't added any users to your private repositories, you should not be concerned with public forks as only you know the repo exists. If you want to block public forks of a private or public repo, you can turn off this feature on the repository's **Admin** page in the **Repository details** section.

Issue Trackers and Wikis

Repositories can include wikis and issue trackers. You can set the visibility (public or private) on a wiki or issue tracker independent of whether the corresponding repo is public or private.

For wikis, the following table describes the various settings you can set and what behaviors the settings enable:

Wiki	Repo	Behavior
public	private	Only users who have access to the private repo, can edit the wiki. Other bitbucket users and any Internet browser can access the Wiki if you publish the URL.

private	private	Only bitbucket users with access can view and edit the wiki.
public	public	Any bitbucket user can search for the repo and then edit the Wiki. Other bitbucket users and any Internet browser can also edit the wiki if you publish the URL.
private	public	Only bitbucket users with access can view and edit the wiki.

For issue trackers, the following table illustrates how you can set an issue tracker's visibility and what behaviors the settings enable.

Issue Tracker	Repo	Behavior
public	private	Only users who have access to the private repo, can create an issue. Other bitbucket users and any Internet browser can view the issue tracker if you publish the URL.
private	private	Only bitbucket users with access to the repo can view, create, and update issues.
public	public	Any bitbucket user can search for the repo and then view or create issues. Other bitbucket users and any Internet browser can also view, create, and update issues if you publish the URL.
private	public	Only bitbucket users with access to the repo can view, create, and update issues.

Regardless of whether an issue tracker is public or private, only a user with **admin** rights can configure a repo's issue tracker.

RELATED TOPICS

Administrating a repository
Managing the Users for your bitbucket Account
Making your bitbucket Repository Private or Public
Making your bitbucket Wiki Private or Public
Making your bitbucket Issues Private or Public

Managing your account

You can use the bitbucket **Account** screen (see screenshot below) to set various options for your bitbucket user account and to modify the way you access bitbucket.

Modifying your Account Settings

To modify your account settings, click **Account** in the top menu bar. Follow these guidelines:

- Managing Repository Users
- Managing the user groups for your Repositories
- Setting Your Email Preferences
- Deleting Your Account
- Renaming Your Account
- Using your Own bitbucket Domain Name

Screenshot: bitbucket account settings

Bitbucket Documentation

RELATED TOPICS

Administering bitbucket

Managing Repository Users

Once you have signed up for a bitbucket username, you automatically have a bitbucket account. If you own one or more private repositories, then your account plan determines the number of users allowed to access those repositories. On the bitbucket **Account** screen, you can see all the users who have access to your private repositories. You can also perform bulk user editing. This is particularly useful if you need to reduce the number of users that count towards your plan.

On this page:

- Overview of User Counts and Repositories in an Account
- Notes

Overview of User Counts and Repositories in an Account

> ✅ **Quick guide to user counts**
>
> A **private** repository may have a restricted number of users, based on the owner's bitbucket plan.
> A **public** repository has an unlimited number of readers, writers and administrators.

bitbucket plans and pricing for private repositories are based on the number of users allowed for the repository owner's account (username).

- Anyone can create an account (username) on bitbucket. Once you have created an account, you are a bitbucket user and an account holder.
- Any bitbucket user can create an unlimited number of public repositories, for no charge. Each public repository can have an unlimited number of users, for no charge.
- Any bitbucket user can create an unlimited number of private repositories for no charge, but the number of users may be restricted. Each account (username) is allowed a maximum number of users, depending on the plan chosen. See bitbucket plans.
- Every repository has a single owner, and that owner's account has a restricted number of users as described above. If the user count reaches the maximum, administrators will not be able to grant repository access to anyone else. Note that each user may have access to one or more than one of the owner's repositories. The count is of individual (distinct) users, not of the number of repository memberships.
- All repositories have unlimited storage.

Example of User Count for an Account

Let's assume that Sarah has a bitbucket account. She owns two private repositories:

- SarahTestPrivate
- SarahTestPrivate2

Sarah also owns a public repository:

- SarahTestPublic

Don, Susan, Peter and Paul have bitbucket accounts.

- Don is a writer in SarahTestPrivate.
- Susan is a reader in SarahTestPrivate and in SarahTestPrivate2.
- Peter and Paul are writers in SarahTestPublic.

Sarah's user count is **3** (Don, Susan and Sarah herself).

*Screenshot: Example of user count as shown on the owner's **'Account'** screen*

Notes

- **Administrators can also add users to repositories.** If you give people administrator permission to your repository, they can add and remove users via the repository **Admin** tab. When administrators are giving a new user access to a private repository, bitbucket

will check the owner's user count. If the additional user takes the count over the account limit, the new user will be rejected.

- **You can organise your users into groups.** You can set up groups for your bitbucket account, to help you to manage user access to your repositories. This is particularly useful if you want to provide similar access to different repositories for the same set of users. For example, let's say you want to provide read-only access to all of your repositories for external contractors. You can set up a group named 'Contractors', assign the usernames for the contractors to the group, and grant this group read-only access to your repositories.

- **What happens when the maximum number of users is exceeded.** If the user count goes over the limit for an account, then access becomes read only for all private repositories for that account. Note that the repository owner still has writer access to each repository, and administrators can still administer the repository. For example, they can remove excess users in order to reduce the user count.

- **What happens when a plan is downgraded.** If a plan is downgraded (which can happen automatically if the account holder does not pay for a renewal) the user count may exceed the limit. Read-only access will apply to all private repositories, as described in the point above, until the user count is within the limit for the account plan.

RELATED TOPICS

Granting Users Access to a bitbucket Repository
Managing the Groups for your bitbucket Account

Managing the user groups for your Repositories

You can create groups to manage user access to your repositories. You create a group at the account level and assign the group to one or more repositories. Creating groups is particularly useful if you want to provide similar access to different repositories for the same set of users.

When you create a group, you can assign it no permissions (**none**) or one of the following default permissions levels:

Level	This permissions allows a user to do the following:
read	View the repository contents. All public repositories grant all bitbucket users read permissions automatically.
write	Contribute to the repository by pushing changes directly from a repository on a local machine.
admin	Do everything a repository owner can do, except delete the repository. This means administrators can: • Change repository settings. • Add, change, and remove user permissions. • Give other users administrator access.

When you create a repository, bitbucket checks to see if you have any groups with a default permissions (**read**, **write**, or **admin**) assigned. If you do, it adds that group to the new repository. If you specify **none** for the default permissions, bitbucket ignores that group. If you are concerned about inadvertently giving user access to your repositories, always specify **none** for the default group permissions. If you change your mind, you can always later go to the repository's **Admin** page and add a group or change a group's permissions.

Keep in mind the group permission applies to all members. If you give a group administrator permission to your repository, any member can add and remove groups via the repository's **Admin** tab. For information about assigning groups permissions at the repository level, see Granting Groups Access to a bitbucket Repository.

The user account for your plan includes the group members. If your user count goes over your plan limit, you can remove individual users from the group or the individual repositories until your back within you plan limits.

Example of creating a group

You create groups from your **Account** page. The following video illustrates adding a group and removing a group:

For example, let's say you want to provide read-only access to all of your repositories for external contractors. To do this you would:

1. Log in to bitbucket and click **Account** in the top menu bar.
2. Click **Groups** in the sidebar.
3. Click **Add group** to create your first group.
   The system displays the **Add a group** dialog.
4. Enter `Contractors` for the group name and click **Add group**.
   The system creates the `Contractors` group with **none** as the default permissions.
5. Click **read** to assign it as the default access.
6. Enter one or more usernames in the field provided.
   When you type in the field, the system attempts to auto complete the name for you.
7. Click **Add user**.
8. Repeat steps 5-6 for each user you want to add.

Related topics

bitbucket 101

Setting Your Email Preferences

bitbucket lets you associate multiple email addresses with your account. These email addresses work site-wide. So, if your local DVCS username/email configuration uses any address on this list, bitbucket automatically associates those with your account. Other bitbucket users can find your account by any registered email address when searching for accounts or users.

You designate one address is the primary account address. bitbucket sends email notifications to this address. Your primary email address is also used to pull your Gravatar icon into your profile. You can change your primary address by clicking **set as primary** on any of your confirmed email addresses.

To map your username to a commit, push, and other activities, bitbucket requires that the email address you commit with matches a validated email address for your account. Both Git and Mercurial allow you to configure a global username/email and a repository specific username/email. If you do not specify a repository specific username/email values, both systems use the global default. So, for example, if your bitbucket account has a validated email address of `joe.foot@gmail.com`, you need to make sure your repository configuration is set for that username. Also, make sure you have set your global username/email address configuration to a validated email address.

If the global default is not configured or if you have not validated your email address, the committer appears as **unknown** for your bitbucket activities. Also, if you have multiple bitbucket accounts, you may mistakenly commit using configuration (global or specific) that maps to an account name you did not intend. To have the existing commits map to a different account, you can use **Admin > Username aliases** for the repository in question. Aliases are per-repository. To set up an alias for a repo, you must have **admin** rights on it.

For information on configuring your local repository or configuring aliases, see Setting your username for bitbucket actions.

Confirming email addresses

When you register a new email address with your account, bitbucket sends you an email containing a confirmation link. Until you confirm that address, bitbucket cannot display your account name with your commits properly and other users cannot find you by that email address. Otherwise, bitbucket performs as it normally would. If you have not confirmed your email address, a banner appears across the top of bitbucket when log in:
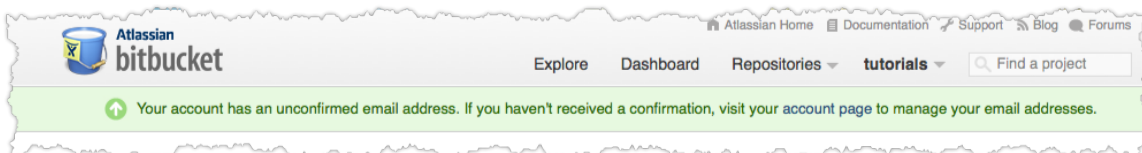
If you've misplaced or your confirmation has expired, do the following:

1. Go to **username > Account > Email addresses**.
2. Click the **resend confirmation** link next to the address.
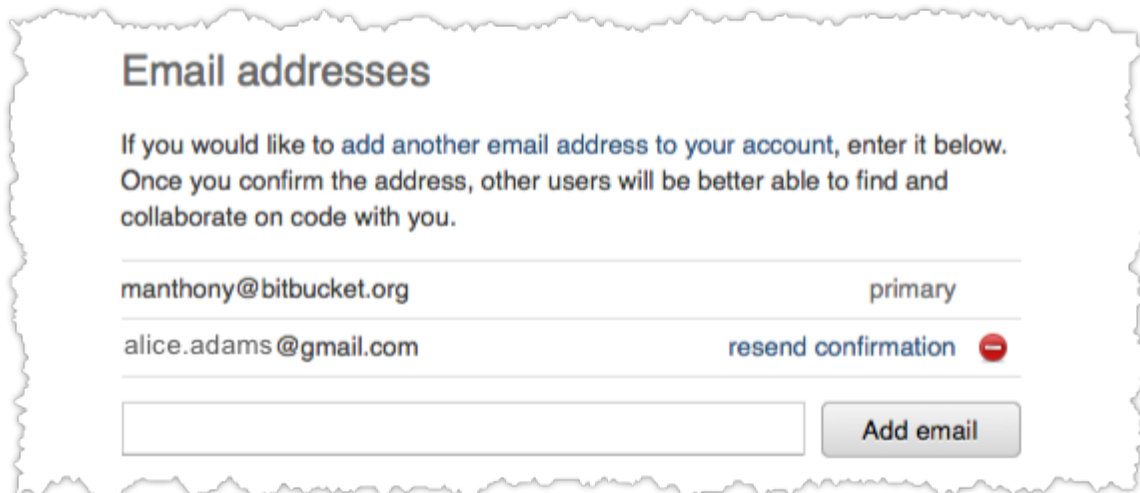
Setting your email notification preferences

Each bitbucket user can choose whether they receive email notifications or not. Your email settings affect email notifications only. Even if you turn off email notifications, you still receive the messages sent to the account **Inbox**. bitbucket sends notifications in the following situations:

Notification	Trigger
Issue tracker notifications	When user adds or updates an issue. These notifications never go to your inbox.
Repository notifications	A user forks a repo, creates a pull request against a repo, gives you access to a repo, or revokes your access.

General notifications	Messages from other users.

To set your preferences for everything but the issue tracker, do the following:

1. Click *username* **> Account** in the top menu bar.
   The **Account settings** page appears.
2. Check or uncheck the **Email notifications** option.
3. Click **Save settings**.

Issue tracker notifications are set per repository. Only a user with admin permissions can set the notifications for an issue tracker. To set permissions for the issue tracker, do the following:

1. Go to the bitbucket **Admin** tab for the relevant repository.
2. Select **Notifications** from the sidebar on the left.
3. Add one or more email addresses to the list.

Deleting Your Account

You can delete your bitbucket account by selecting the **Delete account** option on the bitbucket **Account** screen.

Deleting your bitbucket account will permanently delete all repositories that you have created. As a consequence, any repository URL referencing your account will no longer work.

Screenshot: The delete account option on the Account screen
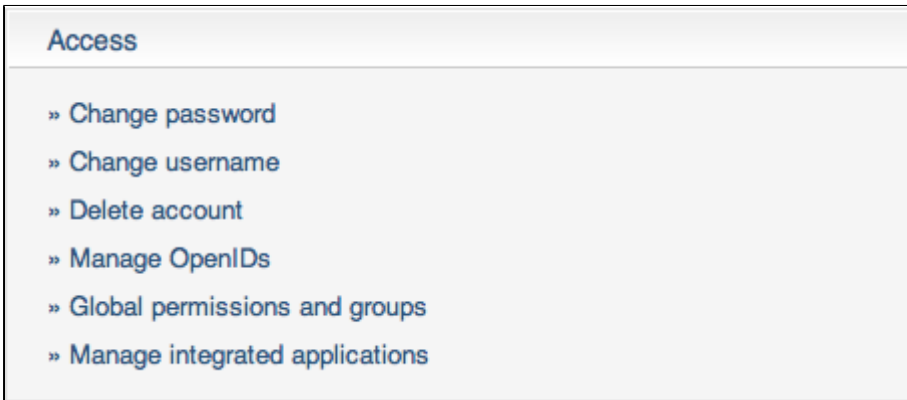
Screenshot: Confirmation message before your account is deleted
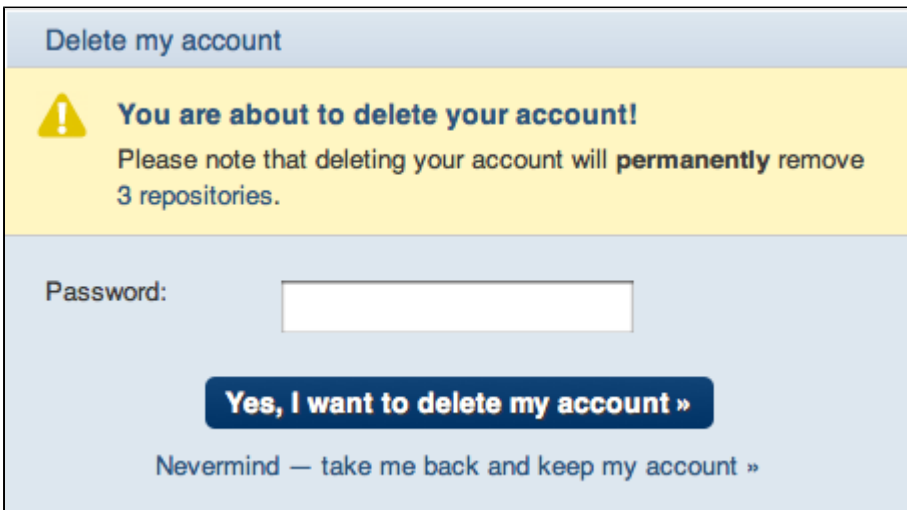
RELATED TOPICS

Renaming Your Account

Renaming Your Account

You can rename your bitbucket account by selecting the **Change username** option on the bitbucket **Account** screen. Renaming your account is the same thing as changing your username.

Changing your username will affect every URL that references your repository and will invalidate every existing URL pointing to your repositories. For example, if you rename your account from *johnc* to *jcitizen*, the repository *repo*, previously available at

`http://bitbucket.org/johnc/repo`, should now be accessed as `http://bitbucket.org/jcitizen/repo`.

Therefore, you, and anyone who has directly cloned your repository, must update the `[paths]` section in every affected repository's configuration file to reflect your new username.

Screenshot: The change username option on the Account screen

Access

» Change password

» Change username

» Delete account

» Manage OpenIDs

» Global permissions and groups

» Manage integrated applications
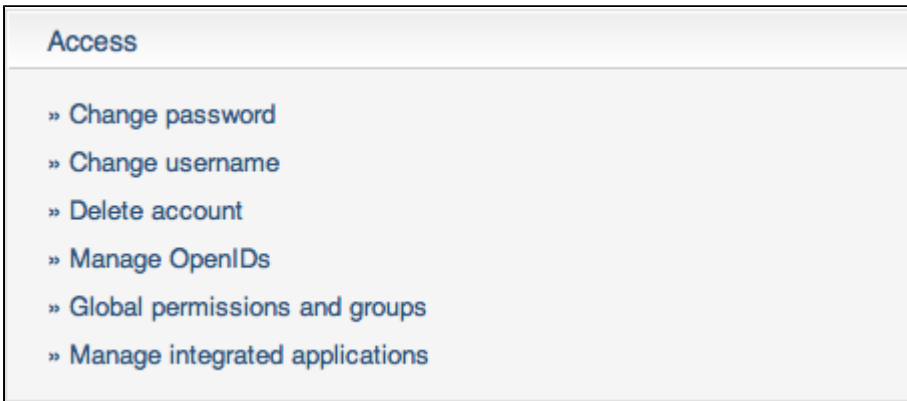
Screenshot: Renaming your account

Change username

Take me back to my account »

Existing username

alui

New username *(required)*

Password *(required)*

⚠ Renaming your account will permanently move the URLs of your 3 repositories.

Rename

RELATED TOPICS

Managing your account
Deleting Your Account

Using your Own bitbucket Domain Name

You can associate a domain with your account. For an example, see http://hg.basho.com/. This option of using a custom domain is available for all bitbucket repositories and for all pricing plans.

To use your own domain name for accessing bitbucket (for example, hg.example.com), you can set up a CNAME.

To set up a CNAME, you must have access to the DNS of your domain name. The actual steps to do this will vary depending on the DNS provider you use. In your domain control panel or configuration, set up the domain name you want as a CNAME pointing to bitbucket.org. For example, you might set up hg.example.com as a CNAME to bitbucket.org.

Then enter this domain name in your bitbucket account settings, and once the DNS change has propagated across the Internet, your new CNAME will be in effect.

HTTPS and CNAMEs

Please note that accessing a CNAME over HTTPS may result in security warnings from your browser.

bitbucket can only serve one SSL certificate, and that certificate can only be associated with the bitbucket.org domain. When you access a

CNAME over HTTPS, your browser will see that the certificate's domain doesn't match the address you've visited and present a warning message.

For this reason, we allow non-secure HTTP access to CNAMEs. However, if you don't mind the certificate warnings and you'd prefer to access your CNAME over an encrypted session, you can still use HTTPS.

For cloning, pushing, and pulling to repositories on CNAMEs, we recommend connecting directly to bitbucket.org over HTTPS or SSH.

Administrating a repository

If you have administrator permissions on a bitbucket repository, you will have access to the **Admin** tab (see screenshot below) which gives access to various administrative functions for the repository.

Getting Administrator Permissions

You can become an administrator of a repository like this:
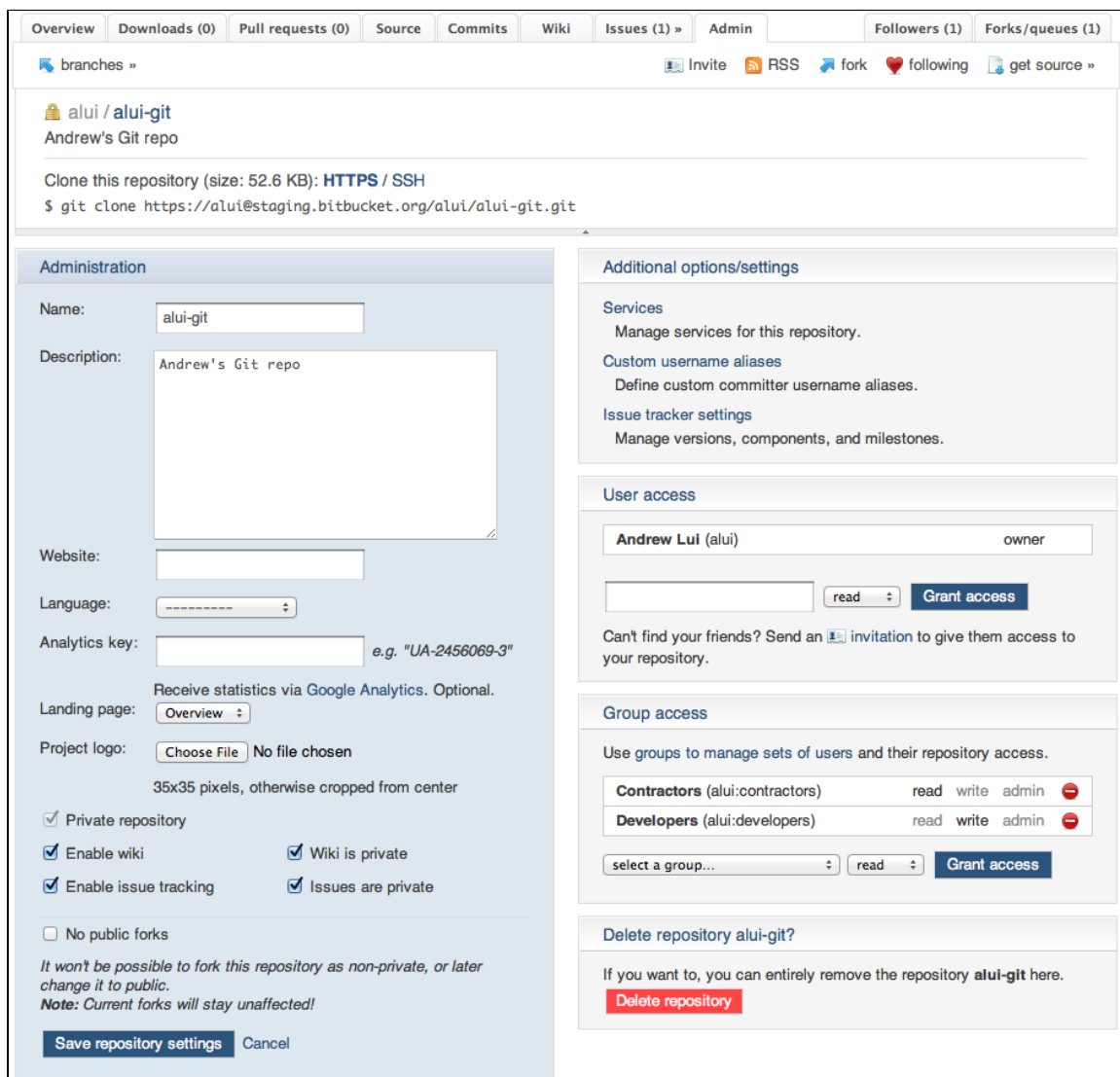
- When you create a bitbucket repository, you are automatically the owner and administrator of that repository.
- Another administrator can add you as an administrator of a repository.

See the overview of bitbucket permissions.

Things an Administrator Can Do

No content found for label(s) bb-admin.

Screenshot: The bitbucket Admin tab

RELATED TOPICS

Managing your account

Granting Groups Access to a Repository

You can set up groups for your bitbucket account, to help you to manage user access to your repositories. This is particularly useful if you want to provide similar access to different repositories for the same set of users. For example, let's say you want to provide read-only access to all of your repositories for external contractors. You can set up a group named 'Contractors', assign the usernames for the contractors to the group, and grant this group read-only access to your repositories.

Administrators can give groups permission to access a bitbucket repository, based on the permission types (privileges) in the table below. When you create a new repository, all groups will be given access to the repository based on the default access for the group. For example, let's say you create a group 'Contractors' with the default access set to 'Read'. When you create a new repository, the 'Contractors' group will be assigned 'Read' access to the new repository. You can change the group's level of access to the repository later.

Notes:

- Any administrator of a repository can grant reader, writer and administrator permissions to any groups for that repository. You do not need to be a repository owner in order to grant permissions.
- Repository owners can add and manage groups via the **Accounts** screen. See Managing the user groups for your Repositories.
- If you are creating/updating a private repository, make sure that you do not exceed the user count for your account.

On this page:

- Administrator Permission is Required
- Permissions that You can Grant
- Granting a Group Access to a Repository
- Number of Users Allowed Per Account

Administrator Permission is Required

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Permissions that You can Grant

Level	This permissions allows a user to do the following:
read	View the repository contents. All public repositories grant all bitbucket users read permissions automatically.
write	Contribute to the repository by pushing changes directly from a repository on a local machine.
admin	Do everything a repository owner can do, except delete the repository. This means administrators can: • Change repository settings. • Add, change, and remove user permissions. • Give other users administrator access.

For more details, see the overview of bitbucket permissions.

Granting a Group Access to a Repository

1. Log in to bitbucket and go to your repository, or create a new one.
2. Click the **Admin** tab for your repository. Below is an explanation of the options on the Group access section of the screen.
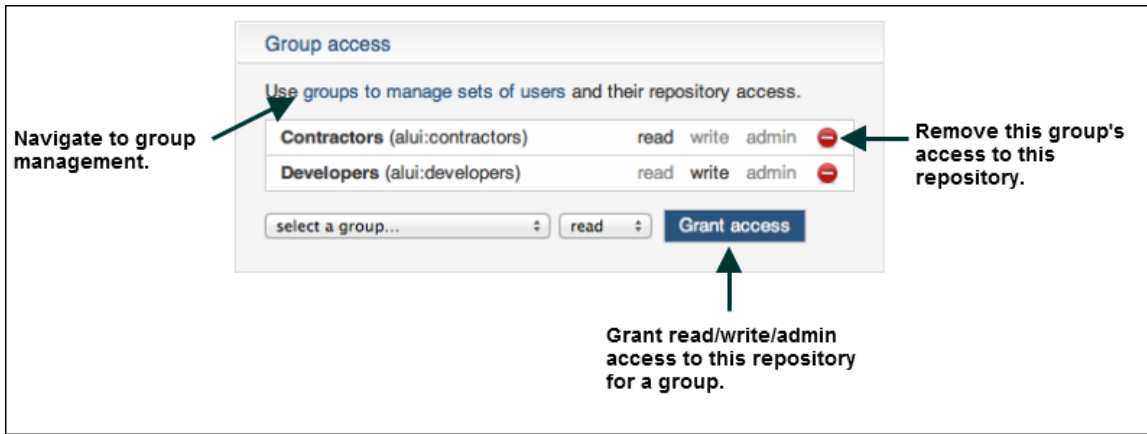
Diagram above: Managing group access to a repository

Number of Users Allowed Per Account

What happens when the maximum number of users for an account is exceeded because you have granted repository permissions to a group? If the user count goes over the limit for your account, then access becomes read only for all your private repositories. Note that the repository owner still has writer access to each repository, and administrators can still administer the repository. For example, they can remove excess users in order to reduce the user count.

RELATED TOPICS

Repository privacy, permissions, and more
Managing the user groups for your Repositories

Granting Users Access to a Repository

Administrators can give other users permission to access a bitbucket repository, based on the permission types (privileges) in the table below.

Notes:

- Any administrator of a repository can grant reader, writer and administrator permissions to any other user for that repository. You do not need to be a repository owner in order to grant permissions.
- The person must already have a bitbucket username, so that you can grant them access to a repository.
- Repository owners can also grant permissions via the **Accounts** screen. See Managing Repository Users.

On this page:

- Administrator Permission is Required
- Permissions that You can Grant
- Granting a User Access to a Repository
- Changing or Revoking a User's Access to a Repository
- Number of Users Allowed Per Account

Administrator Permission is Required

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Permissions that You can Grant

Level	This permissions allows a user to do the following:
read	View the repository contents. All public repositories grant all bitbucket users read permissions automatically.
write	Contribute to the repository by pushing changes directly from a repository on a local machine.
admin	Do everything a repository owner can do, except delete the repository. This means administrators can: • Change repository settings. • Add, change, and remove user permissions. • Give other users administrator access.

For more details, see the overview of bitbucket permissions.

Granting a User Access to a Repository

1. Go to the **Admin** tab for the bitbucket repository.
2. Find the section entitled **User access**.
3. Enter a bitbucket username in the text box. (Start typing, and bitbucket will supply suggested usernames in a dropdown list. Use your mouse or the arrows on your keyboard to select a username.)
4. Select the appropriate permission from the dropdown box.
5. Click **Grant access**

Screenshot: Granting bitbucket repository permissions
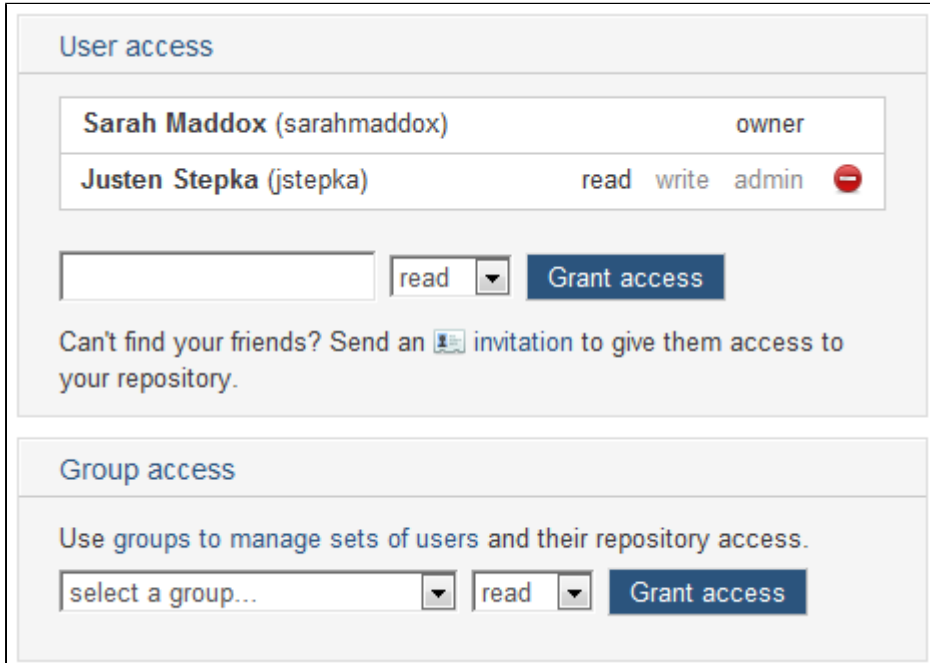
Changing or Revoking a User's Access to a Repository

1. Go to the **Admin** tab for the bitbucket repository.
2. Find the section entitled **User access**.
3. Find the user in the list.
4. Change or delete the user's access:

   - If you want to prevent the user from accessing the repository, click the delete icon.
   - If you want to change the user's access, click the permission (**read**, **write** or **admin**) that you want to assign.

Number of Users Allowed Per Account

✅ **Quick guide to user counts**

A **private** repository may have a restricted number of users, based on the owner's bitbucket plan.
A **public** repository has an unlimited number of readers, writers and administrators.

bitbucket plans and pricing for private repositories are based on the number of users allowed for the repository owner's account (username).

- Anyone can create an account (username) on bitbucket. Once you have created an account, you are a bitbucket user and an account holder.
- Any bitbucket user can create an unlimited number of public repositories, for no charge. Each public repository can have an unlimited number of users, for no charge.
- Any bitbucket user can create an unlimited number of private repositories for no charge, but the number of users may be restricted. Each account (username) is allowed a maximum number of users, depending on the plan chosen. See bitbucket plans.
- Every repository has a single owner, and that owner's account has a restricted number of users as described above. If the user count reaches the maximum, administrators will not be able to grant repository access to anyone else. Note that each user may have access to one or more than one of the owner's repositories. The count is of individual (distinct) users, not of the number of repository memberships.
- All repositories have unlimited storage.

Example of User Count for an Account

Let's assume that Sarah has a bitbucket account. She owns two private repositories:

- SarahTestPrivate
- SarahTestPrivate2

Sarah also owns a public repository:

- SarahTestPublic

Don, Susan, Peter and Paul have bitbucket accounts.

- Don is a writer in SarahTestPrivate.
- Susan is a reader in SarahTestPrivate and in SarahTestPrivate2.
- Peter and Paul are writers in SarahTestPublic.

Sarah's user count is **3** (Don, Susan and Sarah herself).

Screenshot: Example of user count as shown on the owner's **'Account'** screen

RELATED TOPICS

Repository privacy, permissions, and more
Granting Groups Access to a Repository

Managing bitbucket Services

bitbucket offers integration with external services via a set of service brokers that run when certain events happen. These brokers are simple Python scripts, that receive information about the event and then take action. For example, there is a broker for sending email messages, another for posting information to an arbitrary URL, and another for posting updates to a Twitter account.

On this page:

- When Does the Service Run?
- Writing your Own Brokers
- Setting Up your Services
- Available Services

When Does the Service Run?

A service runs once bitbucket receives data from you and other contributors to your project. Because of the decentralised nature of Mercurial, you can commit something locally without notifying bitbucket. It is not until you push your changesets to bitbucket, that you're actually

connecting to our servers and uploading data. As a result, a service may receive more than one changeset. Some of the services take this into consideration. For example, the email broker constructs different email messages based on the number of changesets.

Writing your Own Brokers

You may have special cases, or you may need a broker that we have not yet provided. We have opened up the API as we use for writing the brokers to you. This allows you to write a broker and submit it to us for verification. Once verified, we can put your broker into production and you can start using it.

See our developer guide to writing a broker.

Setting Up your Services

The repository administrator can enable services for a specific repository. Once you've configured a service, it will be active.

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

1. Go to the **Admin** tab for the repository.
2. Click **Services** in the **Additional options/settings** section on the right-hand side of the screen.
3. On the page, you will see a dropdown list of services available. You may add as many services as you want. You can even add many instances of the same type. For example, you can email several people or ping several different URLs.
   - Select the service you want.
   - Click **Add service**.
   - Enter additional information as prompted. Details are in the separate sections per service, listed below.
   - Click **Save settings**.

Screenshot: Adding a service to bitbucket

Available Services

Follow the detailed instructions to set up the services you want:

- Setting Up the Bitbucket Basecamp Service
- Setting Up the Bitbucket CIA.vc Service
- Setting Up the Bitbucket Email Diff Service
- Setting Up the Bitbucket Email Service
- Setting Up the Bitbucket Flowdock Service
- Setting Up the Bitbucket FogBugz Service
- Setting Up the Bitbucket FriendFeed Service
- Setting Up the Bitbucket Geocommit Service
- Setting Up the Bitbucket HipChat Service
- Setting Up the Bitbucket Issues Service
- Setting Up the Bitbucket Jenkins Service
- Setting Up the Bitbucket Lighthouse Service
- Setting Up the Bitbucket Masterbranch Service
- Setting Up the Bitbucket POST Service
- Setting Up the Bitbucket Rietveld Service
- Setting Up the Bitbucket Superfeedr Service
- Setting Up the Bitbucket Twitter Service

Setting Up the Bitbucket Basecamp Service

With this service, you can integrate Bitbucket with Basecamp, a web-based project management application.

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Setting Up the Service

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**Basecamp**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. A new section appears for the Basecamp service. Enter the following information:
   - **Password** – Enter your Basecamp password.
   - **Username** – Enter your Basecamp username.
   - **Discussion URL** – Enter the URL to which Bitbucket should send its update messages.
6. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket CIA.vc Service

With this service, you can integrate Bitbucket with CIA.vc, is an online service which collects change information from version control systems around the world. These changes are broadcast in real time to the website and to Internet Relay Chat.

The primary CIA.vc installation lives at http://cia.vc/.

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Setting Up the Service

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**CIA.vc**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. A new section appears for the CIA.vc service. Enter the following information:
   - **Module** – Enter your CIA.vc module.
   - **Project** – Enter your CIA.vc project.
6. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket Email Diff Service

With this service, you can set up Bitbucket to send an email message to a given email address when the repository is updated. The email service sends an email with diffs.

Setting Up the Service

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**Email Diff**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. Enter the email address where Bitbucket should send the email messages.
6. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket Email Service

With this service, you can set up Bitbucket to send an email message to a given email address, for each pushed changegroup.

Format of the Email Messages

Email messages are sent from `commits-noreply@bitbucket.org`.

When a changegroup contains only one changeset, the email subject line will be of the following form:

```
Subject: commit/<project>: <user>: <commit msg>
```

When a changegroup contains two or more changesets, the email subject line will be of the following form:

```
Subject: commit/<project>: <x> new changesets
```

Setting Up the Service

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**Email**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. Enter the email address where Bitbucket should send the email messages.
6. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket Flowdock Service

With this service, ywhen you push to a repository, Bitbucket will notify Flowdock of the push. Flowdoc then lets you comment on the commit details.

Setting Up the Service

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**Flowdock**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. Enter the required field token which you attain from Flowdock and optionally any tag details you would like to include.
6. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket FogBugz Service

With this service, you can integrate Bitbucket with FogBugz, an issue-tracking and project-management system.

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Setting Up the Service

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**FogBugz**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. A new section appears for the FogBugz service. Enter the URL to which Bitbucket should send its update messages.
6. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket

Managing bitbucket Services

Setting Up the Bitbucket FriendFeed Service

You can add a FriendFeed service to Bitbucket. Bitbucket will post information to FriendFeed as a feed when your project is updated.

Choosing the Format of the FriendFeed Update

The default format is:

```
${repository.name} - ${commit.author}
```

You can change the template to suit your own needs. The fields correspond directly to how the payload is delivered. For a description of the payload, see our guide to writing brokers.

The commit message will be feed's first comment on FriendFeed.

There is also a shortcut for creating a TinyURL to the commit, but note that FriendFeed trims links by default anyway:
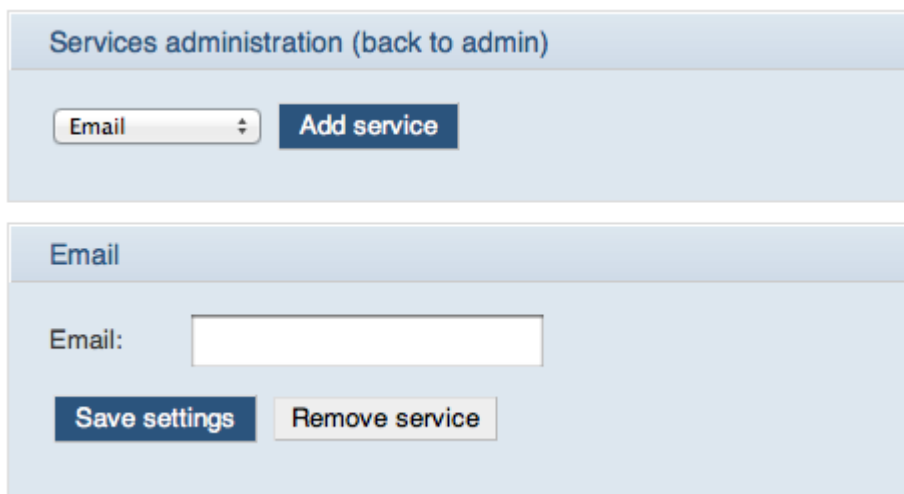
```
${commit.tinyurl}
```

Setting Up the Service

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**FriendFeed**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. A new section appears for the FriendFeed service. Enter the values as follows:
    - **Format** – This is the format of the message sent to FriendFeed. Use the default value or change the template to suit your needs, as described above.
    - **Remote Key** – You can find your remote key at friendfeed.com/remotekey.
    - **Username** – Enter your FriendFeed username.
6. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket Geocommit Service

With this service, you can integrate Bitbucket with Geocommit, a service to add geolocation to your commits.

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Setting Up the Service

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**Geocommit**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket
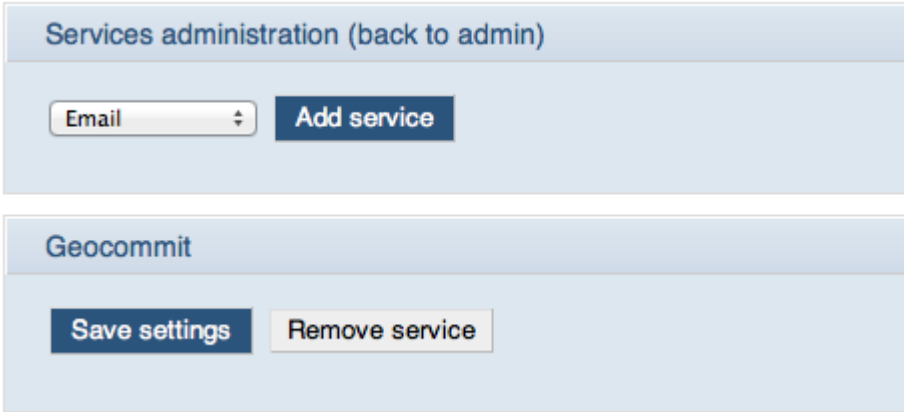
Setting Up the Bitbucket HipChat Service

With this service, you can integrate Bitbucket with HipChat.

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Setting Up the Service

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**HipChat**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. A new section appears for the HipChat service. Enter the following information:#* {*}
    - **Token** -- Enter your HipChat API Token.
    - **Room ID** – Enter your HipChat Room ID.
    - **API URL --** Enter the HipChat API URL. Usually 'http://api.hipchat.com/v1/rooms/message'.
6. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket
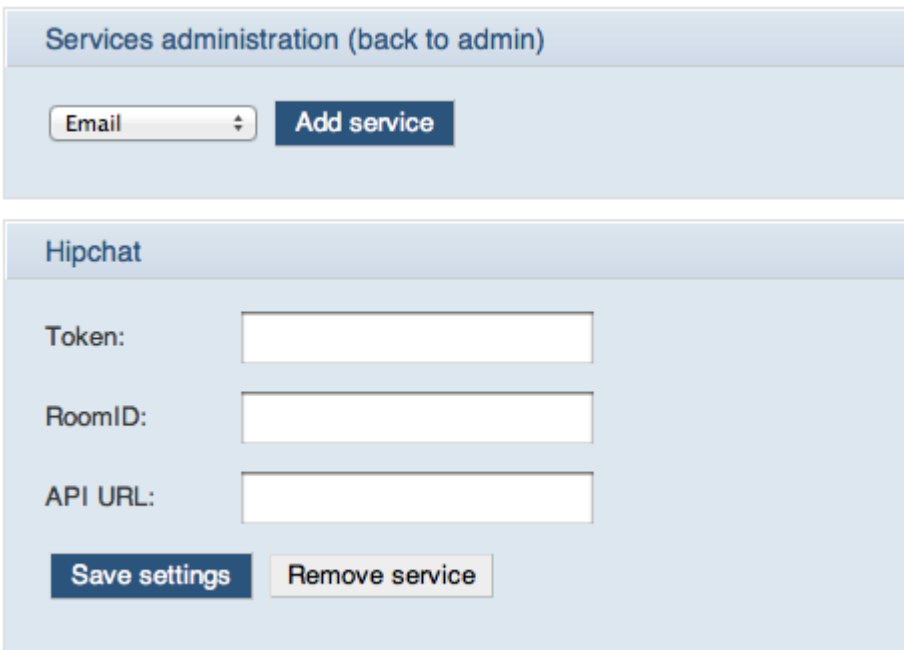
RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket Issues Service

With this service, you can set up Bitbucket to change the state of a given issue, based on the commit message in an update. The Issues service scans commit messages for commands which will automatically change the state of the relevant issue on the Bitbucket issue tracker.

Putting the Issue Information into your Commit Message

To link your commit to an issue, include the following fragment somewhere in your commit message:

```
<command> <issue id>
```

The `<command>` can be any of the following:

```
close/closed/closes/closing/fix/fixed/fixes    # resolves the issue
reopen/reopens/reopening                         # reopens the issue
addresses/re/references/ref/refs/see             # adds a link to the changeset as a comment for
the issue
```

The `<issue id>` can be of the following form, for an example issue 4711:

```
#4711
issue 4711
bug 4711
ticket 4711
```

Examples:

```
"... fixes #4711 ..."                   # marks issue as resolved
   "... reopening bug 4711..."            # marks issue as open
   "... refs ticket 4711..."              # links changeset to issue as comment
   "... refs bug #4711 and #4712..."      # links to multiple issues
```

Note: When displayed in the changeset view, `#4711` will link back to the issue, while `issue 4711` will not.

Setting Up the Service

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

1. Go to the bitbucket repository **Admin** tab.
2. Click **Services** on the left-hand side of the screen.
3. Select the **Issues** service from the dropdown list.
4. Click **Add service**. That's it – done.

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket Jenkins Service

With this service, you can integrate Bitbucket with Jenkins, an extendable open source continuous integration server

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Setting up the service

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**Jenkins**' service from the dropdown list in the '**Services Administration**' section.

4. Click '**Add service**'.
5. A new section appears for the Jenkins service. Enter the following information:\
   - **Endpoint** – Enter your Jenkins endpoint URL.
   - **Project name** – Enter your Jenkins project name.
   - **Token --** Enter the Jenkins token *(optional)*.
   - **Module name --** Enter the Jenkins module name *(optional)*.
6. Click '**Save settings**'.

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket Lighthouse Service

With this service, you can integrate Bitbucket with Lighthouse, a simple issue tracking system.

Setting Up your Project at Lighthouse

1. Create a project in Lighthouse corresponding to the project you are tracking at BitBucket.
2. Access your Lighthouse profile page by clicking on your name next to the 'sign out' link.
3. Select an account from the '**Create an API Token**' dropdown menu on the right-hand side of the page.
4. Give your token a label (anything you want).
5. Make sure you give it '**Full access**' to the project you want to integrate.
6. Click the '**Create**' button.
7. Copy the API token created (just the token).

Setting Up the Service at Bitbucket

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**Lighthouse**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. A new section appears for the Lighthouse service. Enter the following information:
   - **Project ID** – Enter the numeric ID for your project. You can get it from your project URL at Lighthouse.
     For example, let's say your Lighthouse project URL is this:
     XXXXX.lighthouseapp.com/projects/**00000**-yourproject/overview
     In this case, **00000** would be your Project ID.
   - **API Key** – Enter the API token you have created at Lighthouse (see above).
   - **Subdomain** – Enter your Lighthouse account name.
     For example, let's say your Lighthouse project URL is this:
     XXXXX.lighthouseapp.com/projects/**00000**-yourproject/overview
     In this case, **XXXXX** would be your subdomain.
6. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket

Posting Changes to Lighthouse

Now you can easily associate your commits to tickets at Lighthouse, using a special syntax in your commit messages. Here are some examples.

Let's say you have a ticket at lighthouse regarding a login problem and this ticket has ID '22'.

When you commit changes regarding this ticket, you may use:

```
hg ci -m "Commit regarding that login problem. [#22]"
```

This will post this commit as a changeset inside your ticket #22.

Let's say you have resolved that ticket:

```
hg ci -m "Login problem resolved. [#22 state:resolved]"
```

This will automatically change the state of that ticket at Lighthouse.

You can chain many options in your commit, like this:

```
    [#22 state:hold responsible:paul milestone:'My Milestone Name' tagged:nonspacedtag tagged:'a
    spaced tag' tagged:'multiple tags']
```

Summary of the options available:

- **state**: Changes the ticket state.

```
    [#22 state:resolved]
```

- **responsible**: Changes the responsible person for this ticket.

```
    [#22 responsible:lighthouse-username]
```

- **milestone**: Sets this ticket into a milestone.

```
    [#22 milestone:'My Milestone']
```

- **tagged**: Adds tags to the ticket. May be used multiple times.

```
    [#22 tagged:nonspacedtag tagged:'a tag with spaces']
```

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket Masterbranch Service

With this service, you can integrate Bitbucket with Masterbranch.

Setting Up the Service

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**Masterbranch**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket
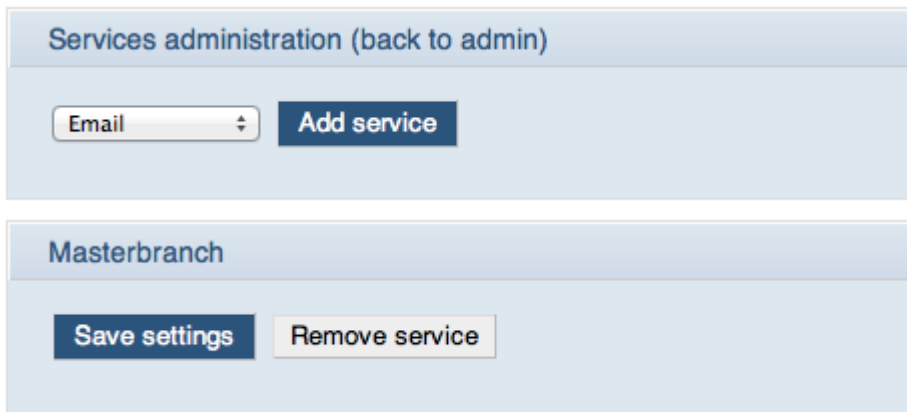
RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket POST Service

You can add a 'POST' service and specify a URL. Bitbucket will post information to the URL when your project is updated. You can think of this as an HTTP publish/subscribe service.

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Setting Up the Service

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**POST**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. A new section appears for the POST service. Enter the URL to which Bitbucket should send its update messages.
6. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket Rietveld Service

With this service, you can integrate Bitbucket with Rietveld, a code review tool.

Setting Up the Service

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**Rietveld**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. A new section appears for the Rietveld service. Enter the following information:
    * **URL**– Enter the URL of your Rietveld site. The default value is:

    ```
    http://reitveld-bitbucket.appspot.com/
    ```

    * **Password** – Enter your Rietveld password.
    * **Email** – Enter your Rietveld email address.
6. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket Superfeedr Service

With this service, you can integrate Bitbucket with Superfeedr.

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Setting Up the Service

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**Superfeedr**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. Click '**Save settings**'.

Screenshot: Adding a service to Bitbucket

RELATED TOPICS

Managing bitbucket Services

Setting Up the Bitbucket Twitter Service

You can add a Twitter service to Bitbucket. Bitbucket will post information to Twitter when your project is updated. Bitbucket will use Twitter's OAuth authentication API.

Choosing the Format of the Twitter Update

The default format is:

```
${repository.name} - ${commit.author} - ${commit.message}
```

You can change the template to suit your own needs. The fields correspond directly to how the payload is delivered. For a description of the payload, see our guide to writing brokers.

There is also a shortcut for creating a TinyURL to the commit:

```
${commit.tinyurl}
```

Setting Up the Service

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

1. Go to the Bitbucket '**Admin**' tab.
2. Click '**Services**' in the '**Additional options/settings**' section on the right-hand side of the screen.
3. Select the '**Twitter**' service from the dropdown list in the '**Services Administration**' section.
4. Click '**Add service**'.
5. A new section appears for the Twitter service. (See screenshot 1 below.) Click the link on the words '**Authorise your Twitter account to communicate with Bitbucket**'.
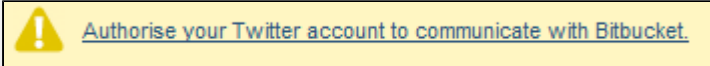
6. A Twitter screen will appear. If you are not logged in to Twitter, it will prompt you to log in now, using your Twitter username and password. If you are already logged in to Twitter, you will go straight to the next step.
7. A Twitter screen will ask if you want to allow or deny Bitbucket access to your Twitter account. (See screenshot 2 below.) Click '**Allow**'.
8. You will be redirected back to Bitbucket. The Twitter '**Secret**' and '**Token**' will be filled in on the Bitbucket service screen. (See screenshot 3 below.)
9. Use the default value for the '**Format**' or change the template to suit your needs, as described above. This is the format of the message sent to Twitter.
10. Click '**Save settings**'.

Screenshot 1: Adding a service to Bitbucket

Screenshot 2: Twitter's OAuth verification screen

Screenshot 3: The Twitter OAuth secret and token on the Bitbucket service screen
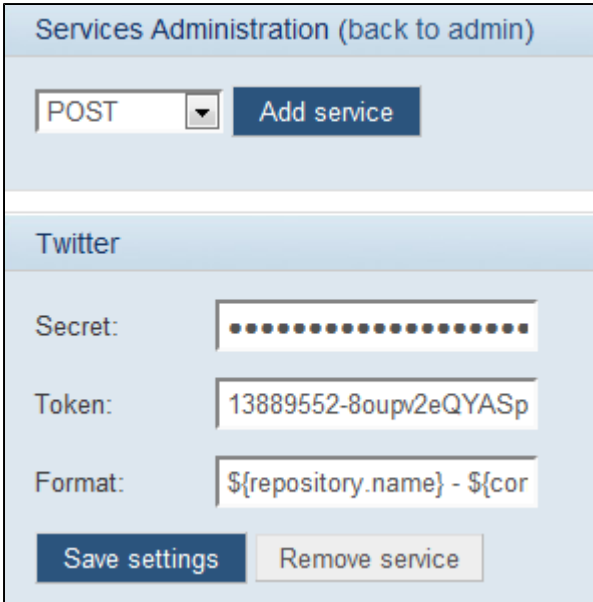
Example of a Twitter Update

Here is an example of a commit update posted to Twitter, using the default message format, for Bitbucket username 'sarahmaddox' and repository 'sarahmaddox', with commit message 'Adding a chocolate picture, of course':

Screenshot 4: A tweet
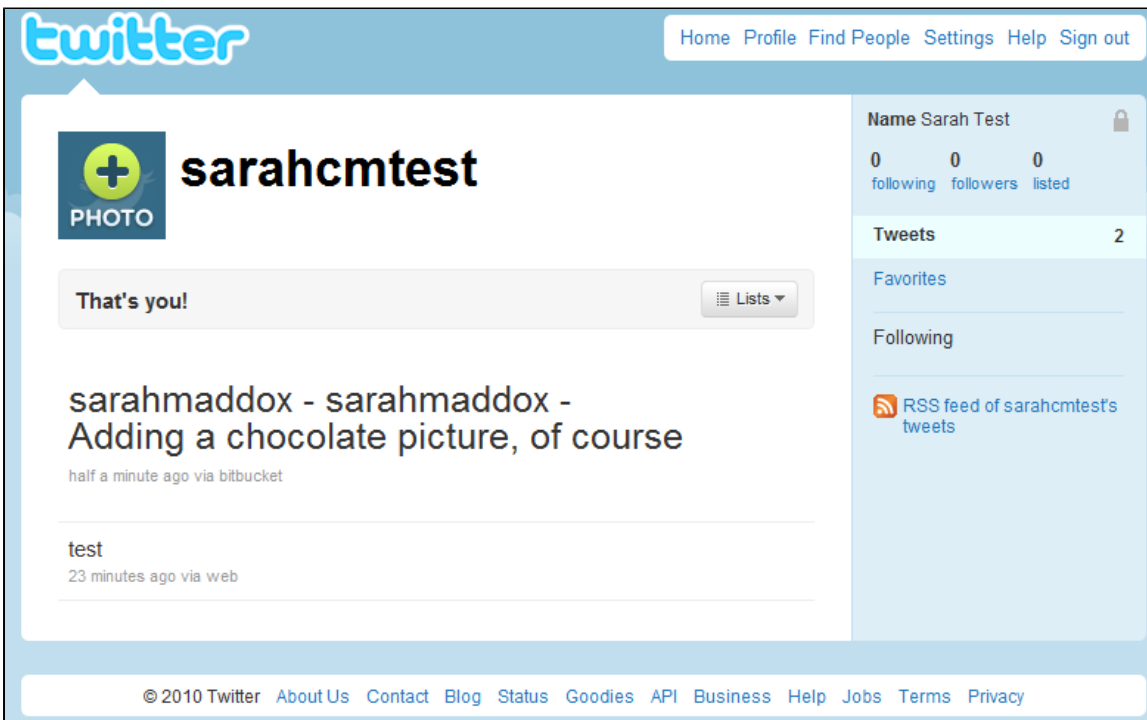
RELATED TOPICS

Managing bitbucket Services

Using your repository

A bitbucket repository is where you can store your code or other project files. bitbucket is great for sharing and collaborating on projects. You can have a private repository, where only the people you choose can see the code. Or you can let everyone see the repository and choose the people who can update it.

- Setting your username for bitbucket actions
- Cloning a Repository
- Converting from Other Version Control Systems
- Forking a bitbucket Repository
- Pushing Updates to a Repository
- Working on Files in your Local Mercurial Repository

- Deleting a Repository
- Branching a Repository
- Making a Repository Private or Public
- Creating a Repository
- Using Patch Queues on bitbucket
- Working with pull requests
- Accessing your bitbucket Repository via Subversion
- Sharing bitbucket Updates with Other Web Services
- Overview - Working on a Copy of a Bitbucket Repository
- Collaborating and Getting Social on bitbucket

RELATED TOPICS

bitbucket 101
Repository privacy, permissions, and more

Setting your username for bitbucket actions

Your commits happen on your local system and you push them to bitbucket. bitbucket uses the commit data to determine which account name to attach to the push in bitbucket. To ensure your actions in bitbucket appear with the correct user, you must configure your local DVCS correctly.

To map your username to a commit, push, and other activities, bitbucket requires that the email address you commit with matches a validated email address for your account. Both Git and Mercurial allow you to configure a global username/email and a repository specific username/email. If you do not specify a repository specific username/email values, both systems use the global default. So, for example, if your bitbucket account has a validated email address of `joe.foot@gmail.com`, you need to make sure your repository configuration is set for that username. Also, make sure you have set your global username/email address configuration to a validated email address.

If the global default is not configured or if you have not validated your email address, the committer appears as **unknown** for your bitbucket activities. Also, if you have multiple bitbucket accounts, you may mistakenly commit using configuration (global or specific) that maps to an account name you did not intend. To have the existing commits map to a different account, you can use **Admin > Username aliases** for the repository in question. Aliases are per-repository. To set up an alias for a repo, you must have **admin** rights on it.

This page contains the following topics:

- Git username/email configuration
- Mercurial username/email configuration
- Alias configuration

Git username/email configuration

Configuring Git works the same across GitBash (Windows) , Mac OSX, and Linux-based systems. To set your global username/email configuration, do the following:

1. Set your username:
   `git config --global user.name "FIRST_NAME LAST_NAME"`
2. Set your email:
   `git config --global user.email "MY_NAME@example.com"`

Configure repository-specific values

To configure repository specific values, do the following:

1. Change directory to the root of the repository.
2. Run the following command:
   `git config user.name "FIRST_NAME LAST_NAME"`
3. Run the following command:
   `git config user.email "MY_NAME@example.com"`
4. Verify your configuration by display your configuration file:
   `cat .git/config`

Mercurial username/email configuration

If you are using TortoiseHg in Windows, do the following to set global repository values:

1. Right-click in your Desktop to open the context menu.
   The system displays the right hand context menu:

2. Click the **Hg Workbench** item.
   The system opens the **TortoiseHg Workbench** application.

3. Choose **File > Settings** to open the **TortoiseHg Settings** dialog.
4. Locate the **Commit** section on the left hand side and click it.
5. Fill in the **Username** value using the following format:
   *firstname lastname***<***email***>**

   When done the dialog looks similar to this:
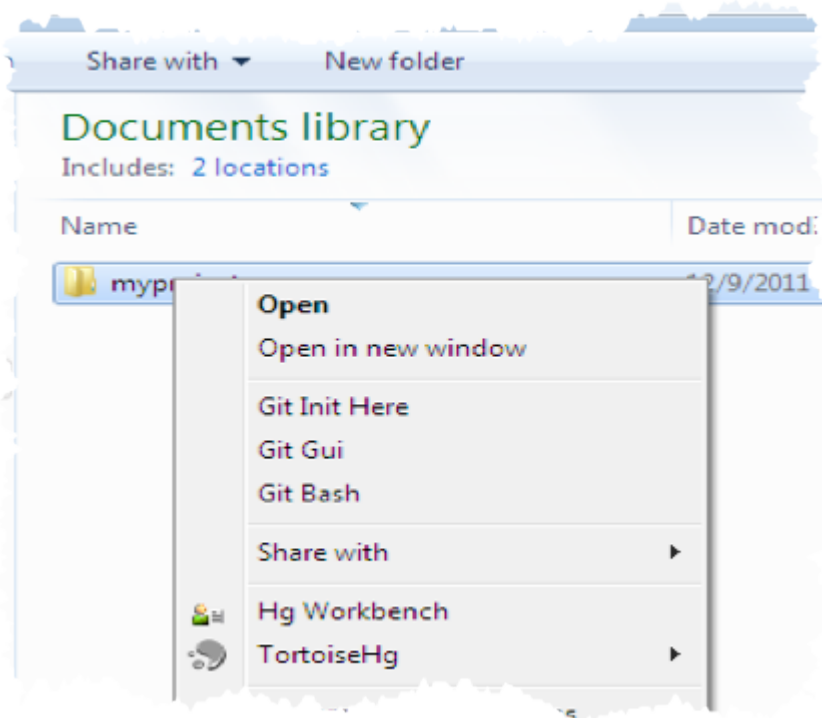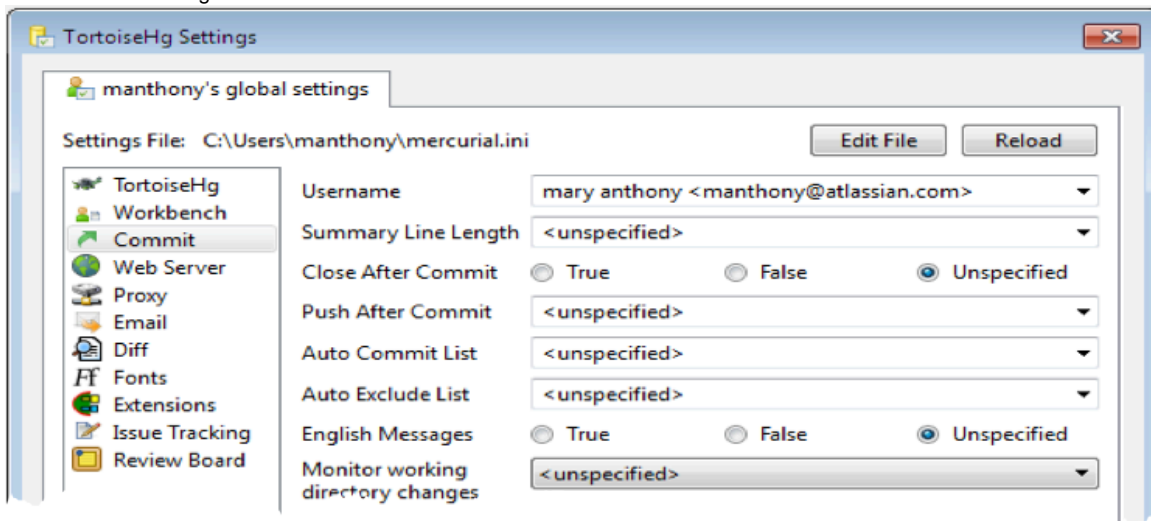
6. Press **OK** to save your changes.

For Mac OSX and Linux systems do the following to set global repository values:

1. Edit the Mercurial configuration file `~/.hgrc` (installed by MacPorts) using your favorite editor.
2. Specify a `username` value.
   When you are done, the configuration file looks something like this:

   ```
   [ui]
   # Name data to appear in commits
   username = Mary Anthony <manthony@atlassian.com>
   ```

3. Save and close the file.

Configure repository-specific values

If you are using TortoiseHg in Windows, do the following:

1. Start TortoiseHG.
2. Right click your repository and choose **Settings**.
   The system displays the **TortoiseHG Settings** dialog with the **NAME repository settings** tab active.
3. Press **Edit File**.
4. View your current repo configuration.

5. Change the `username` value to the value you want to use for that repo.
   When you are done you should see something similar to the following:

   ```
   [ui]
   # Name data to appear in commits
   username = Mary Anthony <manthony@atlassian.com>
   ```

6. Press **Save** to close the editor.

7. Press **OK** to close the settings dialog.

If you are using Mac OSX or a Linux OS, do the following:

1. Open a terminal window.
2. Edit the Mercurial global configuration file (`REPO_INSTALLDIR/.hg/hgrc`).
3. Add the following line to the UI section:

   ```
   ssh = ssh -C
   ```

   When you are done the file should look similar to the following:

   ```
   [ui]
   # Name data to appear in commits
   username = Mary Anthony <manthony@atlassian.com>
   ssh = ssh -C
   ```

4. Save and close the file.

Alias configuration

Only a user with admin permissions for repository can create an alias configuration. An alias is just for that repository; other repositories do not share that alias. Aliases are case sensitive, to alias commits from `foo@bar.com` as well as `FOO@BAR.COM`, you must create two separate aliases. For example, let's assume you used to be `foo@bar.com` and you used that as your DVCS username. Now, you switched to a different email address and updated your bitbucket account accordingly. Commits under the old email address are no longer linked to you:
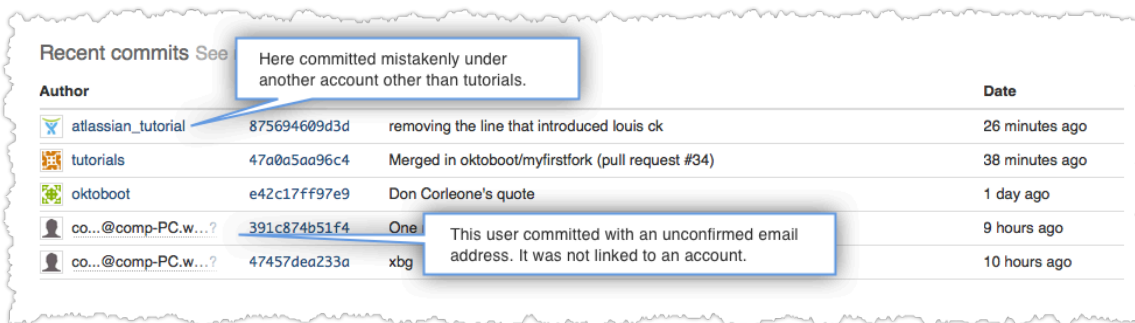
Identify the Author you want to alias and find the author's committer string. To do this:

1. Locate the **Author** in the **Commits** list.
2. Click on the corresponding **Revision**.
   The **Commit** panel appears.
3. Click on the **Raw commit** link.
   The raw commit string passed to bitbucket displays.
4. Locate the **User** section and select the information in it.

```
# HG changeset patch
# User Mary Anthony <manthony@atlassian.com>
# Date 1326832272 28800
# Node ID f492e98bd7cfc30db0539efcd5cde23d741b7431
# Parent  68938015f85c444a0c1e348ea7ff214611ae7f97
adding missing image

diff -r 68938015f85c444a0c1e348ea7ff214611ae7f97 -r f492e98bd7cfc30db0539efcd5cde23d741b7431 images/pulp-fiction.jpg
Binary file images/pulp-fiction.jpg has changed
```

5. Copy to your clipboard.

To make sure bitbucket knows that these old commits were yours, go to the repository administration section and create a custom alias that maps `foo@bar.com` to your account.

1. Log in as the repository administrator.
2. Go to the repository where you want to set up an alias.
3. Click **Admin**.
4. Click **Username Aliases**.
5. Enter the **Username** for the bitbucket user you *want* to appear as the Author.
   In this example, I want the **tutorials** user to show up for my other users. So, I enter `tutorials`.

6. Enter the **Alias** for this user.
   In this example, I'll paste the **User** data from the raw commit. When you are done, your alias looks similar to the following:

Username aliases

For usernames that are mismatched or not found when associating commits with Bitbucket users, specify username aliases.

Username (required)	**Alias** (required)	
tutorials	thony@atlassian.com>	Add alias

7. Press **Add alias**.

After this alias configuration, bitbucket correctly links your commit for this repository:

	addiply	d9e478e99cee	adding quote by Dalai Lama	1 day ago
	tutorials	875694609d3d	removing the line that introduced louis ck	1 day ago
	tutorials	47a0a5aa96c4	Merged in oktoboot/myfirstfork (pull request #34)	1 day ago
	oktoboot	e42c17ff97e9	Don Corleone's quote	2 days ago

Cloning a Repository

When you want to work on a project by changing its files or adding new files, you need to make a local copy of the project onto your machine or local network. To make a local copy of a bitbucket repository, you must clone the repository.

Cloning a Mercurial repository

1. Open a command window and go to the local directory (on your computer) where you want to create your repository.
2. Clone the repository.

```
hg clone https://bitbucket.org/MY_USER/MY_REPO
```

Example:

```
D:\Atlassian\bitbucketTesting\bitbucketRepository>hg clone
https://bitbucket.org/sarahmaddox/sarahmaddox
destination directory: sarahmaddox
no changes found
updating to branch default
0 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

3. A new sub-directory appears, with the same name as the bitbucket repository that you have cloned. The sub-directory contains a folder called `.hg`. That is your Mercurial repository, containing the files and metadata that Mercurial requires to maintain the list of changes you make. Here is its structure as seen in Windows Explorer:

Cloning a Git repository

1. Open a command window.
2. Go to the local directory (on your computer) where you want to clone your bitbucket repository.
3. Clone the repository.

```
git clone https://bitbucket.org/MY_USER/MY_REPO-git.git
```

Example:

```
$ git clone https://alui@bitbucket.org/alui/alui-git.git
Cloning into alui-git...
Password:
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
```

Hint: Your repository **Overview** page in bitbucket displays a repository-specific clone command that you can copy and paste into your command window.

If you were successful, a new sub-directory appears on your local drive. This directory has the same name as the bitbucket repository that you cloned. The clone contains the files and metadata that Git requires to maintain the changes you make to the source files.
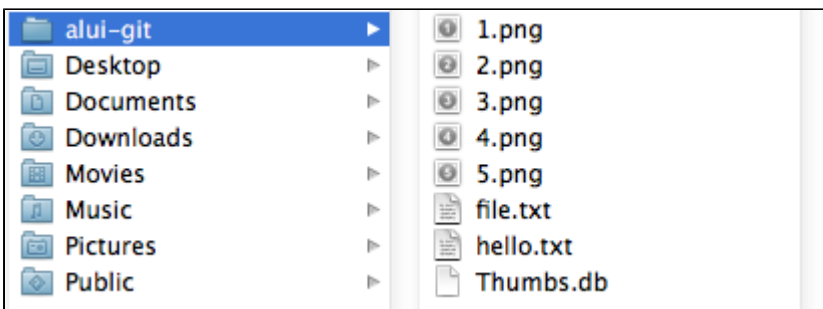
Notes

RELATED TOPICS

Using your repository

Converting from Other Version Control Systems

Conversion to a Mercurial repository is easy. Mercurial ships with a `convert` extension that supports conversion from these systems:

- Bazaar *(experimental, but should work)*
- CVS
- Darcs *(experimental, but should work)*
- Git
- GNU arch *(experimental, but should work)*

- monotone *(experimental, but should work)*
- Subversion (SVN)

Follow our detailed guides for:

- Converting from Subversion to Mercurial

Importing Code

You can import your code into bitbucket without converting it to Mercurial. For instructions, see Importing code from an existing project.

RELATED TOPICS

- An excellent Mercurial manual – enter `hg convert` without specifying any arguments to read the manual
- ConvertExtension on the Mercurial Wiki

Converting from Subversion to Mercurial

This page discusses the options for converting from Subversion to Mercurial.

> If you want to want import a working copy of your code (i.e. no history) from Subversion into Bitbucket, the Bitbucket importer provides a simpler mechanism for doing so. Please see the instructions on Importing Code into your Bitbucket Repository instead.

On this page:

- Option 1: The hgsubversion Extension – Recommended
- Option 2: The convert extension
- Notes

Option 1: The hgsubversion Extension – Recommended

The hgsubversion extension is a user-contributed extension, not distributed with Mercurial. We recommend this extension.

Please refer to the hgsubversion extension guide for download sites and usage instructions.

Option 2: The convert extension

This extension is distributed with Mercurial.

Refer to the documentation:

- Enter `hg convert` without specifying any arguments to read the Mercurial manual.
- Read about the ConvertExtension on the Mercurial Wiki.

Enabling the convert Extension

First of all, you must enable the `convert` extension that ships with Mercurial.

Edit your `~/.hgrc` to look like this:

```
[extensions]
hgext.convert =
```

The `convert` command should now be available.

Subversion

While Subversion is currently used widely in all kinds of software development projects, more and more projects may want to convert to the DVCS paradigm.

For this example, we will be converting Graham Dumpleton's excellent `mod_wsgi` module, which is hosted on Google code.

For Subversion specifically, Mercurial is clever enough to recognize the trunk/branches/tags directory structure used ubiquitously. When you supply an URL for the repository, Mercurial will look for a trunk directory and if it exists, it will use it as the base. If it finds branches or tags it will also attempt to name branches and tags from that.

The Subversion URL for `mod_wsgi` is http:modwsgi.googlecode.com/svn. Lets begin by downloading the latest revision of the repository.

NB: You may also point Mercurial directly at the remote repository, although this is not as well supported as a local starting point. If you want to do this, you can skip this step.

```
$ svn co http://modwsgi.googlecode.com/svn/trunk mod_wsgi
A    mod_wsgi/mod_wsgi
A    mod_wsgi/mod_wsgi/configure
[...]
A    mod_wsgi/README
Checked out revision 929.
```

Now that we have the latest revision of `mod_wsgi` (aka `HEAD`), we have a place we can point Mercurial to. Mercurial will automatically pick up the correct settings and download the full history from the central repository.

```
$ hg convert mod_wsgi mod_wsgi_hg
initializing destination mod_wsgi_hg repository
scanning source...
sorting...
converting...
589 Initial directory structure.
588 Import code from existing private source code repository.
587 Make log message about signals being ignored a warning rather than error.
586 To be consistent with mod_rewrite, %{ENV} should also look in notes and
[...]
0 Fix issue with daemon mode where error logging when using ErrorLog in
updating tags
```

You're done! If you cd to the newly created `mod_wsgi_hg` directory, you will be entering a fully fledged, history-preserved Mercurial repository, consisting of the exact same files as the Subversion repository.

Now would be a good time to upload your repository to Bitbucket, so go ahead and create your repository on the Create Repository page.

Let's upload:

```
$ pwd
~/Work/mod_wsgi_hg
$ hg push http://bitbucket.org/jespern/mod_wsgi
pushing to http://bitbucket.org/jespern/mod_wsgi
searching for changes
remote: adding changesets
remote: adding manifests
remote: adding file changes
remote: added 589 changesets with 1021 changes to 7 files
```

The repository is now uploaded to our servers, and ready to use. Go check it out!

Notes

Related Topics

- Goodbye Subversion, Hello Mercurial: A Migration Guide — Atlassian blog post describing the migration from Subversion to Mercurial.

Forking a bitbucket Repository

In CVS or SVN, you create branches by forking/copying either the head/trunk or an existing branch. In Mercurial, there are various ways to create branches, and there is the concept of forking as described on this page. We recommend forking as the main way for developers to work together.

On this page:

- Overview of Forking
- Step-by-Step Example – First the Fork
- Step-by-Step Example – Now the Pull Request
- Notes
- Screenshots

Overview of Forking

Forking is a way for you to create an exact copy of a repository at a specific point, and take it from there. This is particularly useful if you have

reader permissions for the repository but not writer permissions, and if you want to do some major development work that you may or may not later merge back into the repository.

Here is the basic workflow:

- **Create a fork of the repository.** Go to the repository that you want to fork on bitbucket and click '**fork**'. This gives you a copy of the code from the point where you fork it.
- **Clone your new fork.** Now you may clone your new repository, to create a local copy.
- **Work on the files locally.** Edit the files, add new files, and so on. It is your repository.
- **Commit changesets to your local repository** as usual.
- **Push your changes up to your fork.** Push your changes to bitbucket in the usual way. This will update your fork of the repository.
- **Send a pull request to the owner of the original repository.** Once you are satisfied with your changes, you can ask the owners of the original repository to pull in your changes and merge them into the main repository. Click '**send pull request**'. bitbucket will send a notification to the owner, making it easy for you to communicate with them about your changes from this point onwards.

Step-by-Step Example – First the Fork

Let's go through the steps to fork a repository, make some changes and push them to the new fork. (In a later step, we will ask the repository owners to pull the changes into their repository.)

1. Create a fork of the repository:
   - Go to the repository on bitbucket. For this example, we use a simple test repository https://bitbucket.org/sarahmaddox/sarahmaddox.
   - Click the fork option ⤴ fork on the bitbucket repository screen.
   - The **Fork** options screen appears. See screenshot 1 below. Enter the following information:
   - **Name:** The name of your new repository.
   - **Private:** Select this checkbox if you want your new repository to be private, not public. See Repository privacy, permissions, and more.
   - **Description:** A sentence or two about your new repository. For example, describe why you are forking the repository.
   - **Wiki:** Select this option if you want a bitbucket wiki with your repository. Select **Wiki is private** if you do not want your wiki visible to the general public. See Using your bitbucket Wiki and Making your bitbucket Wiki Private or Public.
   - **Issue tracking:** Select this option if you want a bitbucket issue tracker with your repository. Select **Issues are private** if you do not want your issue tracker visible to the general public. See Using your bitbucket Issue Tracker and Making your bitbucket Issues Private or Public.
   - **Inherit repository user/group permissions:** Select this option if you want the users of the original repository to be able to contribute to your new forked repository by default.
   - **Fork at:** Choose the revision that you want to fork. The default is tip.
   - Click **Fork repository**.
   - bitbucket creates a new repository and opens the overview page of the new repository.
   - You are the owner of this repository. It is a copy of the original, containing the files and metadata of the original. Notice the words **fork of** `original repository name` next to the name of the new repository. See screenshot 2 below.
   - bitbucket sends a message to the owner of the original repository, letting them know that you have forked their repository. See screenshot 3 below. A similar message goes to the owner's email address.
   - Your fork also appears on the **Forks/queues** tab of the original repository. See screenshot 4 below.
2. Clone your new repository (the fork), so that you can start working on it locally. Mercurial and Git examples of this are shown below.
   - Mercurial example:

   ```
   C:\Atlassian\bitbucketTesting\sarahcm-fork>hg clone
   https://bitbucket.org/sarahcm/sarahmaddox-iii
   destination directory: sarahmaddox-iii
   requesting all changes
   adding changesets
   adding manifests
   adding file changes
   added 17 changesets with 29 changes to 18 files
   updating to branch default
   16 files updated, 0 files merged, 0 files removed, 0 files unresolved
   ```

   - Git example:

   ```
   $ git clone https://alui@staging.bitbucket.org/alui/alui-git-fork.git
   Cloning into alui-git-fork...
   Password:
   remote: Counting objects: 3, done.
   remote: Total 3 (delta 0), reused 0 (delta 0)
   Unpacking objects: 100% (3/3), done.
   ```

3. Work on your files locally, as usual. For our example, I edited a text file. I also copied an image file to the local directory, then used the `add`command to add the new image file to the repository.
   - Mercurial example:

```
C:\Atlassian\bitbucketTesting\sarahcm-fork>cd sarahmaddox-iii
C:\Atlassian\bitbucketTesting\sarahcm-fork\sarahmaddox-iii>hg add
adding BalloonTower.png
C:\Atlassian\bitbucketTesting\sarahcm-fork\sarahmaddox-iii>
```

   - Git example:

```
$ cd alui-git-fork
$ ls
BalloonTower.png  file.txt
$ git add *.*
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   BalloonTower.png
#       modified:   file.txt
#
```

4. Commit your changes to the local repository.
   - Mercurial example:

```
C:\Atlassian\bitbucketTesting\sarahcm-fork\sarahmaddox-iii>hg commit -m
"Changed 'odd' to 'occasional' and added pic of balloons"
```

   - Git example:

```
git commit -m "Changed 'foo' to 'I like chocolate.' and added pic of
balloons"
```

5. Push your changes to your repository (the fork) on bitbucket.
   - Mercurial example:
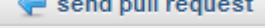
```
C:\Atlassian\bitbucketTesting\sarahcm-fork\sarahmaddox-iii>hg push
pushing to https://bitbucket.org/sarahcm/sarahmaddox-iii
searching for changes
http authorization required
realm: bitbucket.org HTTP
user: sarahcm
password:
remote: adding changesets
remote: adding manifests
remote: adding file changes
remote: added 1 changesets with 2 changes to 2 files
remote: bb/acl: sarahcm is allowed. accepted payload.
```

   - Git example:

```
$ git push
Password:
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 23.98 KiB, done.
Total 4 (delta 0), reused 0 (delta 0)
remote: bb/acl: alui is allowed. accepted payload.
To https://alui@staging.bitbucket.org/alui/alui-git-fork.git
   0ad86b3..395d3c2  master -> master
```

Step-by-Step Example – Now the Pull Request

The steps above showed you how to fork a repository, make some changes and push them to your fork. Now we will ask the owner of the original repository to pull our changes into their repository. We will do this by sending the original owner a pull request.

1. Click the **send pull request** option ⬅ send pull request on your forked repository on bitbucket.
2. The **Send pull request** screen appears. The bottom part of the screen shows the changes made in your fork. The top part of the screen asks for information about your pull request.
3. Scroll down to the **Commits** part of the screen and check the comparison between your fork and the original repository. See screenshot 5 below.
4. Enter the information about your fork and your pull request. See screenshot 6 below.
   - **Pull from:** The branch of your fork that you would like to push into the original repository.
   - **Pull into:** The branch in the original repository where you would like to push your changes to.
   - **Title:** A short description of the changes you have made. This should provide enough information for you and the owners of the other repository to identify your changes.
   - **Description:** More details about your changes.
5. Click **Create pull request**.
   - The pull request appears on the **Pull requests** tab in your fork (see screenshot 7 below) and also in the original repository.
   - bitbucket also sends a message to the owner of the original repository, containing your pull request. See screenshot 8 below.
6. The owner can now reply to your message, examine your changes and decide whether to merge them into the original repository.

Notes

- What happens if you make changes to your fork after sending the pull request? A new option will appear, called **update pull request**. This option is visible to anyone who has writer permission on your fork, and to anyone who has writer permission on the original repository.

- Can you see a diff between your fork and the original repository without sending a pull request? Yes. Click the **compare fork** option.

- Can you see what has changed in the original repository since you made the fork? Yes. Click the **compare fork** option, then click **incoming**.

- What happens if there is a conflict between the two repositories? If someone else has made a change in the original repository that conflicts with your own code:
  - bitbucket will show you the conflicting lines of code. It will also display the code you need to pull the changes and merge them into your repository.
  - You will then resolve the conflicts by comparing the lines of code and fixing them as necessary. Then you will push the changes back to your fork.

Screenshots

Screenshot 1: Fork options

Screenshot 2: A new fork of a repository (Git example)

Screenshot 3: Fork notification (Git example)

Screenshot 4: List of forks (Git example)

Screenshot 5: Creating a pull request (Git example)

Screenshot 7: The pull requests tab on your fork (Git example)

RELATED TOPICS

Working with pull requests
Branching a Repository
Sharing Code in bitbucket

Pushing Updates to a Repository

Once you have changed your code and other files in your local repository, you will need to push them to the remote bitbucket repository so that other people can see them too.

> ⚠️ **Shallow clones unsupported**
>
> bitbucket does not support _shallow clones_. A shallow clone is created by restricting the clone's history to a revision subset. You cannot push a shallow clone to bitibucket.

Permissions

You need writer permission for the repository, to be able to push to the repository. See Repository privacy, permissions, and more.

Pushing your Updates to bitbucket

To push your updates to bitbucket, do the following:

- Mercurial example:

  1. Enter `hg push` at the command line prompt in your local project directory, to copy your files from your local repository to bitbucket.

     ```
     hg push
     ```

  2. When prompted for authentication, enter your bitbucket username and password.
     Example:

     ```
     D:\Atlassian\bitbucketTesting\bitbucketRepository\sarahmaddox>hg push
     pushing to https://bitbucket.org/sarahmaddox/sarahmaddox
     searching for changes
     http authorization required
     realm: bitbucket.org HTTP
     user: sarahmaddox
     password:
     remote: adding changesets
     remote: adding manifests
     remote: adding file changes
     remote: added 2 changesets with 16 changes to 11 files
     remote: bb/acl: sarahmaddox is allowed. accepted payload.
     remote: quota: 108.9 KB in use, 1.0 GB available (0.01% used)
     ```

- Git example:

  1. Enter `git push` at the command line prompt in your local project directory, to copy your files from your local repository to bitbucket.

     ```
     git push
     ```

  2. When prompted for authentication, enter your bitbucket username and password.
     Example:

     ```
     $ git push
     Password:
     Counting objects: 6, done.
     Delta compression using up to 4 threads.
     Compressing objects: 100% (3/3), done.
     Writing objects: 100% (4/4), 23.98 KiB, done.
     Total 4 (delta 0), reused 0 (delta 0)
     remote: bb/acl: alui is allowed. accepted payload.
     To https://alui@staging.bitbucket.org/alui/alui-git-fork.git
        0ad86b3..395d3c2  master -> master
     ```

Open your bitbucket repository in your browser. The **Overview** tab shows your most recent commits:

Screenshot: Recent commits (Git example)

| Overview | Downloads (0) | Pull requests (0) | Source | Commits | Wiki | Issues (1) » | Admin | | Followers (1) | Forks/queues (1) |

 branches »   Invite   RSS   fork   following   get source »

 alui / alui-git
Andrew's Git repo

Clone this repository (size: 52.6 KB): **HTTPS** / SSH
$ git clone https://alui@staging.bitbucket.org/alui/alui-git.git

Recent commits See more »

Author	Message	Date committed	+		−
Andrew Lui	adding flower images	3 minutes ago	5	-	-
Andrew Lui	Adding hello.txt file	4 days ago	1	1	-
Andrew Lui	revising text	7 days ago	1	1	-
Andrew Lui	adding images	8 days ago	5	1	-
Andrew Lui	initial commit	17 days ago	1	-	-

RELATED TOPICS

Cloning a bitbucket Repository
Working on Files in your Local Mercurial Repository

Working on Files in your Local Mercurial Repository

When you want to work on a bitbucket project, you need to make a local copy of the project onto your machine or local network by cloning the repository. See Cloning a Repository. The next step is to work on the code, add files, remove files, and do whatever you want to do.

On this page:

- Working on files in Mercurial
- Working on files in Git
- Notes

Working on files in Mercurial

You will use the Mercurial commands to work on files in your local repository. See the Mercurial documentation for details. Here are the basics:

- **Adding new files:** Add or copy the file to your local project directory, then enter `hg add` to add all new files to the repository at the next commit.
- **Editing files:** Just edit the file in your local project directory and save your changes.
- **Removing files:** Delete the file from your local project directory, then enter `hg remove` to remove the file from the repository at the next commit.
- **Committing your changes to the local repository:** Enter `hg commit` at the command line. To include a commit message, use this syntax:

```
hg commit -m "My commit message."
```

Example – Adding Files to a Mercurial Repository

For our example, let's assume you want to add a number of new files to your repository.

1. Create your new files in your local project directory, or copy the new files from their current location into your local project directory.
2. Enter `hg add` at the command line prompt in your local project directory, to add the files to the repository. Example and results:

```
D:\Atlassian\bitbucketTesting\bitbucketRepository\sarahmaddox>hg add
adding bat.gif
adding feet.jpg
adding foot.jpg
adding horror.gif
adding sint1.jpg
adding sint2.jpg
adding techw_story1 - Copy.htm
adding techw_story1.htm
adding techw_story2.htm
adding techw_story3.htm
adding techw_story6.htm
```

3. Enter `hg commit` at the command line, to commit the new files to the local repository.
4. Mercurial pops up a text editor window, allowing you to enter a message describing the commit.

- Save the text file and exit from the text editor. Mercurial will commit your changes when you exit.
- *Note:*You can use a command line option to specify the commit message, thus avoiding the popup text editor:

```
hg commit -m "Initial commit of all files to the repository."
```

Working on files in Git

You will use the Git commands to work on files in your local repository. See the Git documentation for details. Here are the basics:

- **Adding new files:** Add or copy the file to your local project directory, then enter `git add *.*` to add all new files to the repository at the next commit.
- **Editing files:** Just edit the file in your local project directory and save your changes.
- **Removing files:** Delete the file from your local project directory, then enter `git rm` to remove the file from the repository at the next commit.
- **Committing your changes to the local repository:** Enter `git commit`at the command line. To include a commit message, use this syntax:

```
git commit -m "My commit message."
```

Example – Adding Files to a Git Repository

For our example, let's assume you want to add a number of new files to your repository.

1. Create your new files in your local project directory, or copy the new files from their current location into your local project directory.
2. Enter `git add` at the command line prompt in your local project directory, to add the files to the repository, then enter `git status` to see the changes to be committed.
   Example and results:

```
$ git add *.*
$ git status
# On branch master
# Changes to be committed:
# (use "git reset HEAD <file>..." to unstage)
#
# new file: 1.png
# new file: 2.png
# new file: 3.png
# new file: 4.png
# new file: 5.png
# modified: file.txt
#
```

3. Enter `git commit` at the command line, to commit the new files to the local repository.
4. Git pops up a text editor window (the VI text editor by default), allowing you to enter a message describing the commit.

5. Save the text file (*press Escape to go to command mode, then enter `:wq`*) and exit from the text editor. Git will commit your changes when you exit.

```
$ git commit
[master 2f41d64] adding images
 6 files changed, 1 insertions(+), 1 deletions(-)
 create mode 100644 African Daisy.png
 create mode 100644 Dandelion.png
 create mode 100644 Ixia.png
 create mode 100644 Spiked.png
 create mode 100644 Sunflower.png
```

-   *Note:*You can use a command line option to specify the commit message, thus avoiding the popup text editor:

```
git commit -m "Initial commit of all files to the repository."
```

Notes

RELATED TOPICS

Cloning a Repository
Pushing Updates to a bitbucket Repository
http://mercurial.selenic.com/wiki/GitConcepts — Overview of Git for Mercurial users, including a comparison of Mercurial and Git commands.
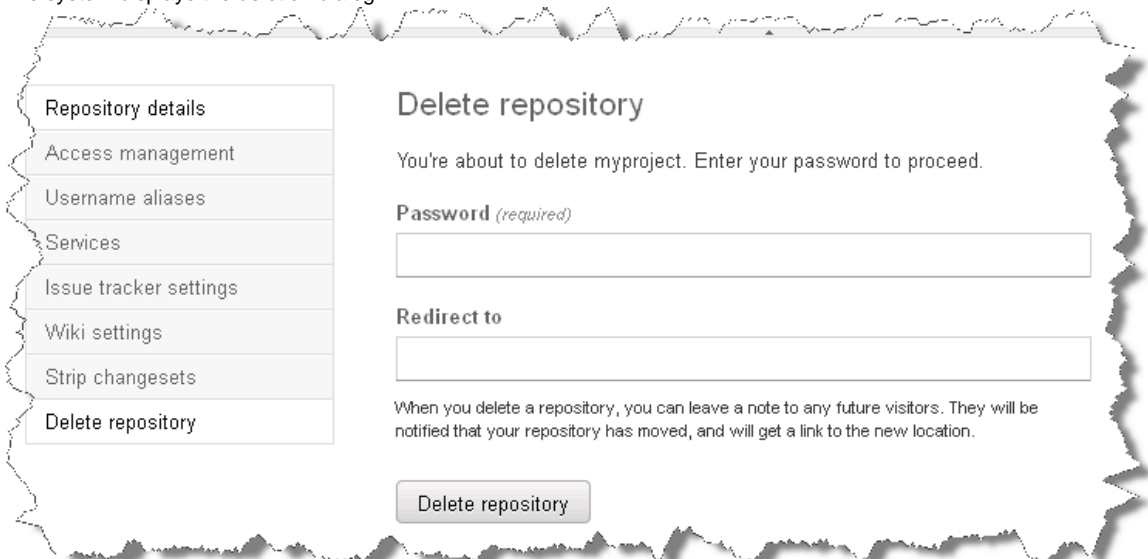
Deleting a Repository

Only the repository owner can delete the repository. The owner is the person who created the repository. Deleting a repository permanently removes the repository from bitbucket. As a consequence, any URL referencing this repository no longer works. To delete a repository:

1. Log into your bitbucket account.
2. Go to the repository's **Admin** tab.
3. Choose Delete Repository from the left-hand navigation.
    The system displays the deletion dialog:

4. Enter your **Password**.

> ℹ If you log into bitbucket with OpenId, you enter the password associated with that id. For example, if you created your bitbucket account through you Google account's OpenID, you enter the password for your Google account.

5. (Optional) Enter a URL in the **Redirect to** field.
    Future users who try to access your repository, are sent to this new URL.

6. Press **Delete repository**.

Branching a Repository

Branching offers a way to work on a new feature without affecting the main code line. You can branch on a local repo and make changes without ever making that branch visible in bitbucket. Or you can push a branch up to bitbucket so other repo users can access and work with your changes. There are a number of ways to branch a repository this page only shows you the very basic ways to branch. Review your Git or Mercurial resources to get fancy with branching. From a bitbucket perspective, you have to do some extra work to get a branch to appear:

Git	Mercurial
Clone a repository to your local system.	Clone a repository to your local system.
Create a branch on your local system. `git branch BRANCH_NAME`	Create a named branch. `hg branch NAME`
Push the branch to bitbucket. `git push origin BRANCH_NAME`	Push the repo to bitbucket. `hg push`

Git branches always have a name. Mercurial has the concept of branch and named branch. (Steve Losh's guide is a good resource explaining branching in the two systems).

On this page:

- Comparing Branching and Forking
- Listing a repo's branches in bitbucket
- How to branch in Git
- How to branch a Mercurial repository
- External Guides to Branching

Comparing Branching and Forking

Forking and branching provide two ways of diverging from the main code line. Both Mercurial and Git have the concept of branches at the local level. The code that is branched and the branch know and really on each other. Like a tree branch, a code branch knows about the tree (original code base) it originated from. The origin of the term fork (in programming) comes from an Unix system call that created a copy of an existing process. A fork is another way of saying clone. As DVCS hosting has evolved, the term fork has grown. The bitbucket software adds management to forks; forking a repo in bitbucket has functionality you normally wouldn't associate with DVCS clone. Whether you use either branching or forking and to what extent depends on your working environment.

There are lots of ways a team can work with and combine fork and branch functionality. You can google for discussions about this. Generally, for hosted systems, forks work well in situations where, as a repo admin, you don't want to manage user access on your repo. Branching works well in situations where you have a small group of programers who trust each other and who all have write access to a repository. The bitbucket team recommends forking rather than branching for these reasons:

- Forking creates an independent repository. These can be simpler to work with.
- If you fork, you are the admin of the repo you create and have full control over it. For example, you can add other users.
- If you need to abandon your changes, it is easier to delete a fork than a branch.

Finally, forks don't require the coordination with the repo owner that branching does. Ultimately, though it is your choice – branch or fork – bitbucket supports both.

Listing a repo's branches in bitbucket

1. Log into bitbucket.
2. Navigate in your browser to the **Overview** page of a repo.
3. Move your mouse over the **branches** value immediately under the first tab.
   The system lists the available branches:
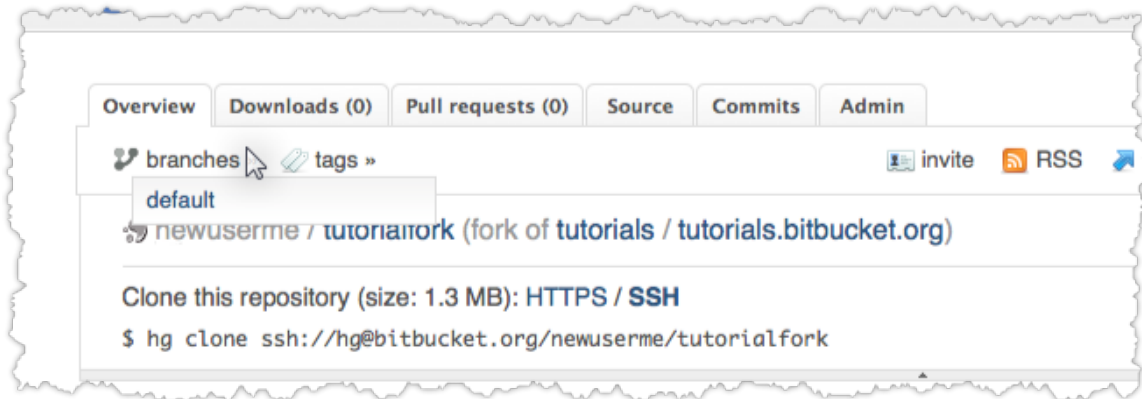
Mercurial repos always have a single `default` branch. Git repos always have a single `master` repo.

How to branch in Git

If you still have your `bb101practice` repo from the 101, you can try these operations directly on that.

1. Open a terminal window on your local system.
2. List the branches on your repo.

```
$ git branch
* master
```

This output indicates there is a single branch, the `master` and the asterisk indicates it is currently active.

3. Create a new `feature` branch in the repository

```
$ git branch feature
```

4. Switch to the `feature` branch to work on it.

```
$ git checkout feature
```

List the branches again with the `git branch` command.

5. Edit the `README` file and add a new line.
6. Commit the change to the `feature` branch:

```
$ git add .
$ git commit -m "adding a change from the feature branch"
```

7. Switch back to the `master` branch.

```
$ git checkout master
```

8. Push the `feature` branch to bitbucket.

```
$ git push origin feature
```

9. View the **bb101repo Overview** page in bitbucket.
   You should see both the master and the feature branch.
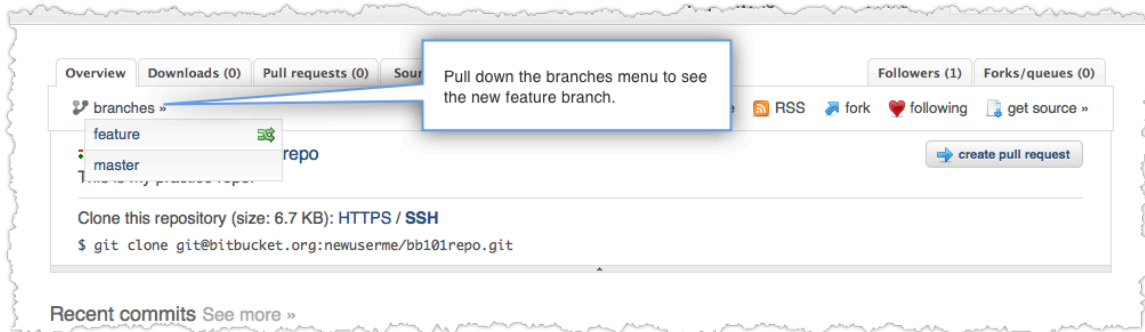
When you select the `feature` branch, you see the **Overview** material from that perspective.

10. Select the `feature` branch to view its **Recent commits**.

How to branch a Mercurial repository

1. Create a new feature branch in the repository.

```
hg branch feature
```

2. Update the working copy to the tip of the feature branch.

```
hg update feature
```

3. Edit files, add files, etc, in the working copy.
4. When you have finished working with the branch, you can merge it back to the main repository. First switch back to the default repository.

```
hg update default
```

5. Merge the feature branch.

```
hg merge feature
```

6. Commit the changes to the repository.

```
hg commit
```

7. Push to bitbucket.

```
hg push
```

External Guides to Branching

Here are some good guides to branching in Mercurial and DVCS:

- The Mercurial documentation.
- A Guide to Branching in Mercurial, by Steve Losh.

Making a Repository Private or Public

You can set your bitbucket repository as private or public.

Public versus private: A private repository is visible only to people who are explicitly given permission to see it. A public repository is visible to everyone. The same applies to the bitbucket issue tracker and wiki, which can be either private or public.

Separate settings for repository, wiki and issue tracker: You can set your bitbucket repository, wiki and issue tracker as private or public,

independently of each other. For example, you can hide your code from the world by setting your repository as private, but let people see your documentation and issues by marking your wiki and issue tracker as public. Or you could set your repository and wiki as public but keep your issue tracker private. And so on.

Swapping the settings around: It is OK change the settings from private to public, or public to private, at any time.

Permissions

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Configuring the Private/Public Settings

1. Go to the bitbucket **Admin** tab.
2. Tick the **Private repository** check box to make your code base private. Untick it to make the code base public.
3. Click **Save repository settings**.

Screenshot: Private/Public Settings on the bitbucket Admin tab
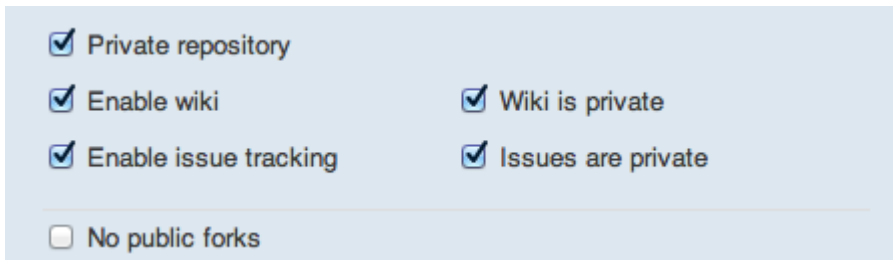
RELATED TOPICS

Repository privacy, permissions, and more

Creating a Repository

This page tells you how to create a new repository in bitbucket. You can create a Mercurial or a Git repository. If you haven't used bitbucket before, we recommend reading bitbucket 101 to get started.

Note, you will need to sign up for an account at bitbucket.org before you can create a repository.

Creating your Repository

1. Click the create new repository button   on your bitbucket home screen.
2. Enter a **name** and **description** for your repository.
3. Tick **Private** if you want to hide your repository from the general public, so that only selected people can see it.
4. Select the Repository type, Mercurial or Git, as desired.
5. Click **Create repository**.

You can also create a new repository by importing code from other code hosting sites or repositories. See _Importing Code into your bitbucket Repository._

Using Patch Queues on bitbucket

This guide is being written - until then, you may read the very useful explanation/guide by Ches :

http://ches.nausicaamedia.com/articles/technogeekery/using-mercurial-queues-and-bitbucket-org

Working with pull requests

This page describes how to work with pull requests. A pull request has source code that comes from a repo (fork or branch) and has an associated revision. When a pull request originates in a branch of your repo, it is commonly called "a pull across branches." You can incorporate a pull request into your repo using bitbucket. You can also incorporate a request by pulling the code directly into your local repo and pushing the repo back to bitbucket.

If a pull request conflicts with your repo, you must pull the code to your local repo for resolving a request. You resolve the conflicts locally by merging or whatever other means works best. Then, push the changes back to bitbucket. After the push, bitbucket shows the request as accepted. Pulling a request locally is also a means for evaluating a code change before accepting the change.

On this page:

- Creating a pull request

-

Creating a pull request

Before creating a pull request, you should compare your outgoing requests to the destination repo. You should also see how the destination repo changed while you worked on your fork or branch. You do this by comparing incoming changes. It is good practice to make sure that there are no incoming changes before you make your pull request. The following video demonstrates how to create a pull request.

Or you can use this procedure.

1. Press **send pull request**.
2. The system displays the request form.
3. Complete the form.

4. Press **create pull request**.
   The system opens your latest request on the **Pull Request** page *of the original repo*.
5. To see the list of all the pull requests against this repo, click the **Pull Request** tab.
   The system also sent a notification email to your account Inbox.
6. Go ahead and open your account inbox *username* **> Inbox**.
7. Locate and review the pull request email (it should be near the top).

Accepting a pull request without Conflicts

Only a user with write permissions on the destination repo can accept or reject a pull request. Any user with read permission in a repository can review the open, accepted and rejected pull requests.

The following procedure illustrates a pull request:

1. Click the **Pull requests** tab in your bitbucket repository.
   A list of incoming pull requests under the heading **Open**.
2. Click the relevant pull request.
   The pull request details appear, including a comparison of the fork and the original repository.
3. Examine the comparison.
4. If you decide that you want to merge the fork into your own repository, click **Accept and pull**.
5. Confirm the action when prompted.
   bitbucket merges the changes into your repository, all on the bitbucket server.

> ℹ️ If an **Update pull request** button appears by a request this means that the fork changed after the pull request was sent. You can click **Update pull request** to apply those changes to the pull request. Or, you can accept and pull what is in the request alone.

Pull requests in local Mercurial repos

This is a simple Mercurial example of the procedure for pulling changes made by another user from the main bitbucket repository, down into the local working copy.

1. Check for incoming changes *(one change detected)*.

```
D:\Atlassian\bitbucketTesting\bitbucketRepository\sarahmaddox>hg incoming
https://bitbucket.org/sarahcm/sarahmaddox
comparing with https://bitbucket.org/sarahcm/sarahmaddox
searching for changes
changeset:   6:9f0a3d22d0be
tag:         tip
user:        Sarah Maddox <smaddox@atlassian.com>
date:        Fri Aug 27 12:07:21 2010 +1000
summary:     Changed 'bounding' to 'pounding' and added pic of cockatoo'
```

2. Check for outgoing changes *(no changes detected)*.

```
D:\Atlassian\bitbucketTesting\bitbucketRepository\sarahmaddox>hg outgoing
https://bitbucket.org/sarahcm/sarahmaddox
comparing with https://bitbucket.org/sarahcm/sarahmaddox
searching for changes
no changes found
```

3. Pull the changes into the local repository.

```
D:\Atlassian\bitbucketTesting\bitbucketRepository\sarahmaddox>hg pull
https://bitbucket.org/sarahcm/sarahmaddox
pulling from https://bitbucket.org/sarahcm/sarahmaddox
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 2 changes to 2 files
(run 'hg update' to get a working copy)
```

4. Update the working copy.

```
D:\Atlassian\bitbucketTesting\bitbucketRepository\sarahmaddox>hg update
2 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

5. Push the changes up to bitbucket.

```
D:\Atlassian\bitbucketTesting\bitbucketRepository\sarahmaddox>hg push
pushing to http://bitbucket.org/sarahmaddox/sarahmaddox
searching for changes
http authorization required
realm: bitbucket.org HTTP
user: sarahmaddox
password:
remote: adding changesets
remote: adding manifests
remote: adding file changes
remote: added 1 changesets with 2 changes to 2 files
```

Pull request in a local Git repository

This is a simple Git example of the procedure for pulling changes made by another user from a fork of a bitbucket repository, back into the original repository also on bitbucket.

1. Check for incoming changes *(one change detected)*.

```
$ git fetch && git log ..origin/master
Password:
commit 2f41d64e75f2b4f0405cbfb1d2f882882809c209
Author: ANDREW <alui@atlassian.com>
Date: Thu Sep 8 11:50:08 2011 +1000

  adding images
```

2. Check for outgoing changes *(no changes detected).*

```
$ git fetch && git log origin/master..
Password:
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From https://staging.bitbucket.org/alui/alui-git
 2f41d64..7e28ddb master -> origin/master
```

3. Pull the changes into the local repository.

```
$ git pull
Password:
Updating 2f41d64..7e28ddb
Fast-forward
 Thumbs.db | Bin 0 -> 11776 bytes
 file.txt | 4 +++-
 2 files changed, 3 insertions(+), 1 deletions(-)
 create mode 100644 Thumbs.db
```

Accessing your bitbucket Repository via Subversion

⚠ **This feature is unsupported and only works with Mercurial (not Git)**

This feature is known to fail and is not supported. If you try it out, we would love to hear your feedback. Please add comments to this page. In particular, this feature generally does not work for large repositories.

You can access bitbucket repositories via Subversion. You can even commit to them. SVN access is enabled for all repositories.

Below we show you how to access bitbucket via Subversion on an ongoing basis. If you want to migrate from SVN to bitbucket, read Converting from Subversion to Mercurial instead.

Structure of your Subversion Repository

bitbucket has a bridge that emulates the conventional structure of Subversion repositories:

- A `trunk` directory to hold the main line of development.
- A `branches` directory to contain branch copies.
- A `tags` directory to contain tag copies.

Checking Out Trunk – Getting Latest from bitbucket

A Subversion checkout gets the latest versions of the files from the bitbucket repository. Checking out from trunk is the equivalent of getting the files from bitbucket's tip.

ℹ It is important to include the word **trunk** at the end of the path, as shown below.

1. Open a command window and go to the directory where you have created your Subversion repository.
2. Enter the Subversion checkout command:

```
svn checkout http://bitbucket.org/MY_USER/MY_REPO/trunk
```

Try it:

```
svn co http://bitbucket.org/jespern/django-piston/trunk
```

Example:

```
C:\Atlassian\bitbucketTesting\BBFromSVN\BBSVNRepo>svn checkout
http://bitbucket.org/sarahmaddox/sarahmaddox
A      sarahmaddox\tags
A      sarahmaddox\branches
A      sarahmaddox\trunk
A      sarahmaddox\trunk\techw_story6.htm
A      sarahmaddox\trunk\foot.jpg
A      sarahmaddox\trunk\bat.gif
A      sarahmaddox\trunk\README
A      sarahmaddox\trunk\techw_story2.htm
A      sarahmaddox\trunk\techw_story3.htm
A      sarahmaddox\trunk\sint1.jpg
A      sarahmaddox\trunk\techw_story1 - Copy.htm
A      sarahmaddox\trunk\horror.gif
A      sarahmaddox\trunk\techw_story1.htm
A      sarahmaddox\trunk\sint2.jpg
A      sarahmaddox\trunk\feet.jpg
A      sarahmaddox\trunk\HotChoc.jpg
Checked out revision 5.
```

Updating your Local Repository – Pulling from bitbucket

A Subversion update will update your local files with all the changes in bitbucket. It is equivalent of a bitbucket pull.

1. Open a command window and go to the directory where your Subversion working copy is stored.
2. Enter the Subversion update command:

```
svn update --username MY_USER
```

Example:

```
C:\Atlassian\bitbucketTesting\BBFromSVN\BBSVNRepo\sarahmaddox>svn update --username sarahmaddox
U    trunk\techw_story3.htm
Updated to revision 6.
```

Committing your Changes – Pushing Changes to bitbucket

A Subversion commit is the equivalent of a bitbucket push. It writes your changes back to the bitbucket repository.

1. Open a command window and go to the directory where your Subversion working copy is stored.
2. Enter the Subversion commit command:

```
svn commit -m "MY COMMIT MESSAGE." --username MY_USER
```

3. When prompted for authentication, enter your bitbucket username and password.

RELATED TOPICS

Converting from Subversion to Mercurial

Sharing bitbucket Updates with Other Web Services

You can configure bitbucket to send a notification to another web service when your repository is updated. If you like, you can send the notification to a number of web services.

bitbucket offers integration with external services by allowing a set of 'brokers' to run at certain events. These brokers are simple Python scripts that receive information about the event and take action. For example, there is a broker for sending email messages, another for posting information to an arbitrary URL, and another for posting updates to a Twitter account.

If you are a bitbucket repository administrator, you can set up these services. See Managing bitbucket Services.

Overview - Working on a Copy of a Bitbucket Repository

There are a number of ways to take a copy of a bitbucket repository so that you can work on the project. Each method is slightly different and is done for different reasons.

Cloning a Repository

This is useful if you have writer permission for the repository.

When you want to work on a project by updating its files or adding new files, you need to make a local copy of the project onto your machine or local network. To make a local copy of a bitbucket repository, you must 'clone' the repository. Read more about cloning a bitbucket repository.

Forking a Repository

Forking is a way for you to create an exact copy of a repository at a specific point, and take it from there. This is particularly useful if you have reader permissions for the repository but not writer permissions, and if you want to do some major development work that you may or may not later merge back into the repository.

Here is the basic workflow:

* **Create a fork of the repository.** Go to the repository that you want to fork on bitbucket and click '**fork**'. This gives you a copy of the code from the point where you fork it.
* **Clone your new fork.** Now you may clone your new repository, to create a local copy.
* **Work on the files locally.** Edit the files, add new files, and so on. It is your repository.
* **Commit changesets to your local repository** as usual.
* **Push your changes up to your fork.** Push your changes to bitbucket in the usual way. This will update your fork of the repository.
* **Send a pull request to the owner of the original repository.** Once you are satisfied with your changes, you can ask the owners of the original repository to pull in your changes and merge them into the main repository. Click '**send pull request**'. bitbucket will send a notification to the owner, making it easy for you to communicate with them about your changes from this point onwards.

Read more about forking a bitbucket repository.

Branching a Repository

Branching offers a way to work on a new feature without affecting the main code line. You can branch on a local repo and make changes without ever making that branch visible in bitbucket. Or you can push a branch up to bitbucket so other repo users can access and work with your changes. There are a number of ways to branch a repository this page only shows you the very basic ways to branch. Review your Git or Mercurial resources to get fancy with branching. From a bitbucket perspective, you have to do some extra work to get a branch to appear:

Git	Mercurial
Clone a repository to your local system.	Clone a repository to your local system.
Create a branch on your local system. `git branch BRANCH_NAME`	Create a named branch. `hg branch NAME`

Push the branch to bitbucket. `git push origin BRANCH_NAME`	Push the repo to bitbucket. `hg push`

Git branches always have a name. Mercurial has the concept of branch and named branch. (Steve Losh's guide is a good resource explaining branching in the two systems).

Read more about branching a bitbucket repository.

A Comparison of Branching and Forking

Forking and branching provide two ways of diverging from the main code line. Both Mercurial and Git have the concept of branches at the local level. The code that is branched and the branch know and really on each other. Like a tree branch, a code branch knows about the tree (original code base) it originated from. The origin of the term fork (in programming) comes from an Unix system call that created a copy of an existing process. A fork is another way of saying clone. As DVCS hosting has evolved, the term fork has grown. The bitbucket software adds management to forks; forking a repo in bitbucket has functionality you normally wouldn't associate with DVCS clone. Whether you use either branching or forking and to what extent depends on your working environment.

There are lots of ways a team can work with and combine fork and branch functionality. You can google for discussions about this. Generally, for hosted systems, forks work well in situations where, as a repo admin, you don't want to manage user access on your repo. Branching works well in situations where you have a small group of programers who trust each other and who all have write access to a repository. The bitbucket team recommends forking rather than branching for these reasons:

- Forking creates an independent repository. These can be simpler to work with.
- If you fork, you are the admin of the repo you create and have full control over it. For example, you can add other users.
- If you need to abandon your changes, it is easier to delete a fork than a branch.

Finally, forks don't require the coordination with the repo owner that branching does. Ultimately, though it is your choice – branch or fork – bitbucket supports both.

Collaborating and Getting Social on bitbucket

Collaborating with other people is very easy and convenient with a DVCS. Bitbucket makes it easier by providing a number of simple yet powerful features.

Finding others with interesting projects

There are a lot of ways to discover new people and code. Log into bitbucket and click **Explore**.

Areas to explore	This shows you
Trending repositories	Public repositories with a lot of activity such as commits, followers, or forks. You can choose what period of time to calculate these over.
Featured repositories	A revolving list of public repositories in bitbucket. We choose these for you.
Blogospherical disturbances	The latest tweets about bitbucket and/or the repositories it hosts. A search service compiles this list. Use the links to navigate to a Twitter feed or to the post.

When you see a ***username / repository*** combination in trending or featured repositories, these are links you can click. Clicking on a username takes you to the user's overview page. From that page you can:

- get more information about a user such as location or website
- browse through a user's public repositories and latest commit activity
- send a user a message
- follow the user's future activities (the heart icon)

Clicking on a repository takes you to the repo overview. Here you can follow the repo (heart icon!) or explore the project in greater depth.

Giving others access to your repo

Sharing your code and collaborating with other people is what bitbucket is all about. bitbucket makes it easy by providing a simple yet powerful access control system. You can choose whether your repository is private or public. Using bitbucket's Access Control Lists (ACLs), you can specify the read and write access to your repository.

- If you have a private repository, you may for example invite your friends and colleagues to view your code by giving them read access, or invite them to contribute changes by giving them write access.
- If your repository is public, you may individually provide write access to whomever you want. This is useful for open source projects where you do not mind everybody having access to read the files, but only want to allow write access to your team.

Read more about bitbucket permissions.

Using your bitbucket Issue Tracker

When you add a repository to bitbucket, you also get an issue tracker. This is the place to track your project's feature requests, bug reports and other project management tasks. We keep the bitbucket issue tracker very simple and yet somewhat flexible. It has just a few configurable fields (component, version and milestone) – you can use them any way you want.

Overview of the Issue Tracker

Screenshot: The bitbucket issue tracker

Issues (1)						
All Open My issues Build query Find issues						Create issue
status: "new" or "open"						
Title	**?**	**!**	**State**	**Responsible**	**Version**	**Date created**
#1: Tech writer review and welcoming image			new	Andrew Lui		11 days ago

Detailed Issue Tracker How To Guides

- Enabling an Issue Tracker
- Making Issues Private or Public
- Adding Components to an Issue Tracker
- Adding Versions to an Issue Tracker
- Adding Milestones to an Issue Tracker
- Adding Spam Prevention to an Issue Tracker
- Automatically Resolving Issues on Push
- Setting Email Notification Preferences for an Issue Tracker
- Using Syntax Highlighting and Markup in the Issue Tracker

Enabling an Issue Tracker

You can choose whether your bitbucket project will include an issue tracker or not.

Permissions

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Enabling the Issue Tracker

1. Go to the **Admin** tab of your bitbucket project.
2. Tick the **Enable issue tracking** check box to bring your issue tracker to life. Untick it to disable the issue tracker.
3. Click **Save repository settings**.

Screenshot: Enabling the issue tracker on the bitbucket Admin tab

☐ Private repository   ☐ Ignore pull requests
☑ Enable wiki   ☐ Wiki is private
☑ Enable issue tracking   ☐ Issues are private

☐ No public forks
It won't be possible to fork this repository as non-private, or later change it to public.
NB: *Current forks will stay unaffected!*

RELATED TOPICS

Making your bitbucket Issues Private or Public

Making Issues Private or Public

You can set your bitbucket issue tracker as private or public.

Public versus private: A private repository is visible only to people who are explicitly given permission to see it. A public repository is visible to everyone. The same applies to the bitbucket issue tracker and wiki, which can be either private or public.

Separate settings for repository, wiki and issue tracker: You can set your bitbucket repository, wiki and issue tracker as private or public, independently of each other. For example, you can hide your code from the world by setting your repository as private, but let people see your documentation and issues by marking your wiki and issue tracker as public. Or you could set your repository and wiki as public but keep your issue tracker private. And so on.

Swapping the settings around: It is OK change the settings from private to public, or public to private, at any time.

Permissions

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Configuring the Private/Public Settings

1. Go to the bitbucket **Admin** tab.
2. Tick the **Issues are private** check box to make your issue tracker private. Untick it to make the issue tracker public.
3. Click **Save repository settings**.

Screenshot: Private/Public Settings on the bitbucket Admin tab


```
☐ Private repository        ☐ Ignore pull requests
☑ Enable wiki               ☐ Wiki is private
☑ Enable issue tracking     ☐ Issues are private

☐ No public forks
It won't be possible to fork this repository as non-private, or later change it to
public.
NB: Current forks will stay unaffected!
```

RELATED TOPICS

Repository privacy, permissions, and more

Adding Components to an Issue Tracker

Components are sub-sections of a project. You can use a component to group issues within a project into logical groups. You can add components to a repository's issue tracker. The components will then be available for selection via a dropdown list when someone creates an issue. You can also set the default component, which will appear as the default selection in the dropdown list.

Examples of component names:

- User Interface
- Installer
- Documentation
- Plugins
- Macros

Permissions

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Adding a Component

1. Go to the bitbucket **Admin** tab.
2. Click **Issue Tracker Settings** near the top right of the screen.
3. Enter the name of the component in the **Components** section.
4. Click **Add component**.

Screenshot: Adding components on the bitbucket Issue Tracker Settings screen
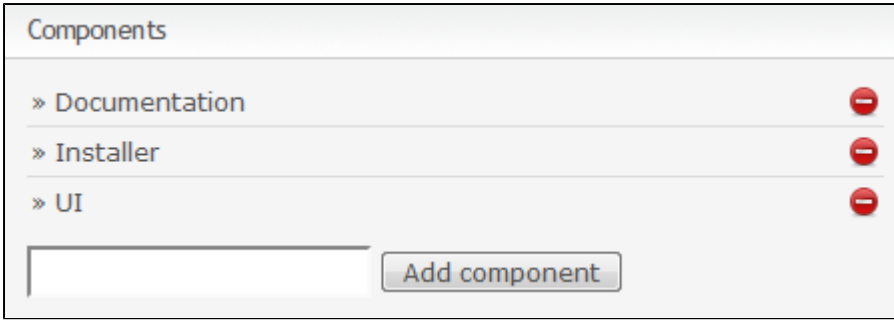
Assigning a Default Component

1. Go to the **Issue Tracker Settings** screen as described above.
2. Select the default component in the section labelled **Set default values for new issues**. The dropdown list will show all the components that have been added to this issue tracker.
3. Click **Save defaults**.

Screenshot: Assigning default components on the bitbucket Issue Tracker Settings screen
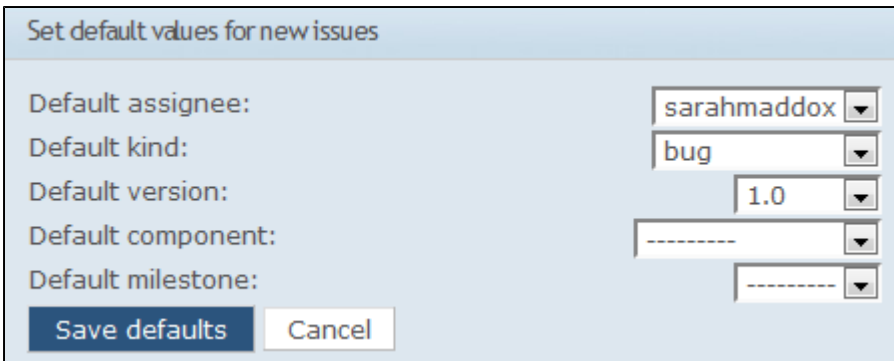
RELATED TOPICS

Using your bitbucket Issue Tracker

Adding Versions to an Issue Tracker

Versions are points in time for a project or product. In a software development project, a version may be the same as a release number. Versions help you schedule and organise your releases and track the release that is affected by a bug.

You can add versions to a repository's issue tracker. The versions will then be available for selection via a dropdown list when someone creates an issue. You can also set the default version, which will appear as the default selection in the dropdown list.

Examples of versions:

- 1.0
- 1.1
- 1.1.1
- 1.1.2
- v1.0
- v1.1
- And so on.

Permissions

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Adding a Version

1. Go to the bitbucket **Admin** tab.
2. Click **Issue Tracker Settings** near the top right of the screen.
3. Enter the version number or name in the **Versions** section.
4. Click **Add version**.

Screenshot: Adding versions on the bitbucket Issue Tracker Settings screen

Assigning a Default Version

1. Go to the **Issue Tracker Settings** screen as described above.
2. Select the default version in the section labelled **Set default values for new issues**. The dropdown list will show all the versions that have been added to this issue tracker.
3. Click **Save defaults**.

Screenshot: Assigning default versions on the bitbucket Issue Tracker Settings screen

RELATED TOPICS

Using your bitbucket Issue Tracker

Adding Milestones to an Issue Tracker

A milestone is usually seen as a subset of a version. It is a point that the development team works towards. It corresponds to a short period of time in which the development team implements and delivers a discrete product increment, such as a working milestone release.

You can add milestones to a repository's issue tracker. The milestones will then be available for selection via a dropdown list when someone creates an issue. You can also set the default milestone, which will appear as the default selection in the dropdown list.

Examples of milestones:

- M1
- M2
- V1.1.m1
- V1.1.m2

Permissions

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Adding a Milestone

1. Go to the Bitbucket **Admin** tab.
2. Click **Issue Tracker Settings** near the top right of the screen.
3. Enter the milestone number or name in the **Milestones** section.
4. Click **Add milestone**.

Screenshot: Adding milestones on the Bitbucket Issue Tracker Settings screen
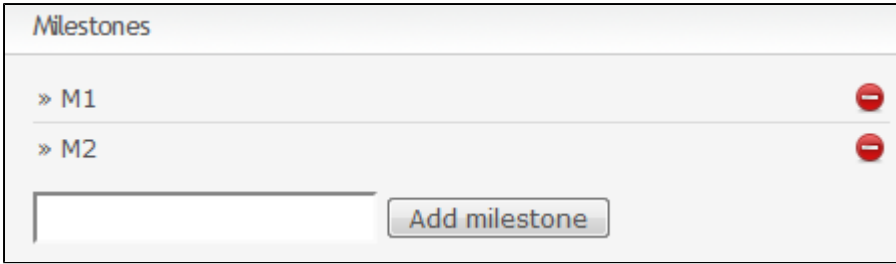
Assigning a Default Milestone

1. Go to the **Issue Tracker Settings** screen as described above.
2. Select the default milestone in the section labelled **Set default values for new issues**. The dropdown list will show all the milestones that have been added to this issue tracker.
3. Click **Save defaults**.

Screenshot: Assigning default milestones on the Bitbucket Issue Tracker Settings screen
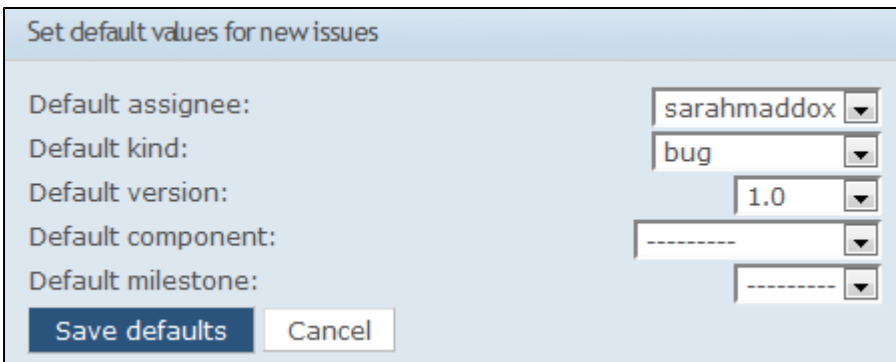
RELATED TOPICS

Using your bitbucket Issue Tracker

Adding Spam Prevention to an Issue Tracker

bitbucket uses an Akismet plugin for spam prevention. If you would like to reduce spam coming into your bitbucket issue tracker, you can get an API key from Akismet and register it with bitbucket, as described below.

Permissions

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Configuring Spam Prevention

1. Go to Akismet and obtain an API key.
2. Go to the bitbucket **Admin** tab.
3. Click **Issue Tracker Settings** near the top right of the screen.
4. Paste your Akismet API key into the **Akismet key** text box.
5. Click **Save Akismet key**.

Screenshot: Configuring spam prevention for your bitbucket issue tracker
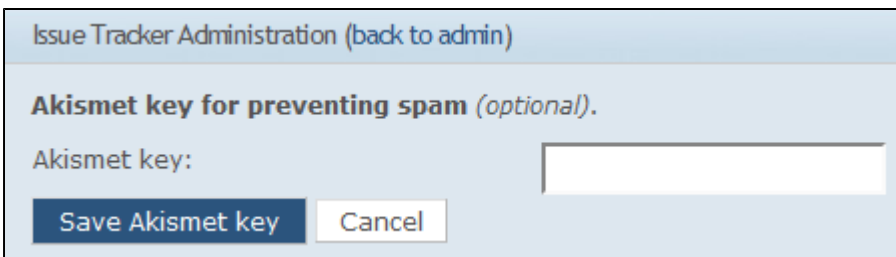
RELATED TOPICS

Using your bitbucket Issue Tracker

Automatically Resolving Issues on Push

You can use bitbucket to resolve tickets automatically when people push updates through to your bitbucket repository. To set this up, configure the appropriate service on the repository. If you want to resolve issues in bitbucket's issue tracker, you must set up the **Issues** service. See Setting Up the bitbucket Issues Service.

Setting Email Notification Preferences for an Issue Tracker

bitbucket can send an email to specified email addresses whenever a user creates new issue or updates an existing one. Issue tracker notifications go to email only; they do not go to an account's **Inbox**. If an user disabled email notifications, the user does not receive issue tracker notifications regardless of how you set a repo's issue tracker notifications.

Issue tracker notifications are set per repository. Only a user with admin permissions can set the notifications for an issue tracker. To set permissions for the issue tracker, do the following:

1. Go to the bitbucket **Admin** tab for the relevant repository.
2. Select **Notifications** from the sidebar on the left.
3. Add one or more email addresses to the list.

RELATED TOPICS

Setting Your Email Preferences
Using your bitbucket Issue Tracker

Using Syntax Highlighting and Markup in the Issue Tracker

Syntax highlighting and markup in the issue tracker are the same as in the bitbucket wiki:

- Using Wiki Markup in bitbucket
- Using Syntax Highlighting on your bitbucket Wiki

Using a bitbucket Wiki

When you add a repository to bitbucket, you also get a wiki. The wiki is a simple place to keep documents. Some people use it as their project home page. For example, take a look at the TortoiseHg wiki on bitbucket.

The wiki is a Mercurial/Git repository, so you can clone it and take it with you. See below for details.

Updating your Wiki Pages Online

1. Go to the bitbucket **Wiki** tab and find the page you want.
2. Click **Edit**.
3. Make your changes to the page content. See Using Wiki Markup in bitbucket.
4. If you want to, you can enter a comment in the **Message** text box. This is a commit message. What you enter here will appear in the page history above the relevant commit entry.
5. Click **Save**.

Screenshot: The Edit link on a bitbucket wiki page

Updating your Wiki Pages Offline, by Cloning the Wiki Repository

The wiki itself is a Mercurial or Git repository, depending on the repository that the wiki is associated with. That means you can clone it, edit it locally or offline, add images or other files, then push it back to bitbucket. Your changes will be live immediately. The repository retains a

history of all updates, whether made online or offline.

To clone your wiki:

```
$ hg clone http://bitbucket.org/MY_USER/MY_REPO/wiki
```

or

```
$ git clone http://bitbucket.org/MY_USER/MY_REPO/wiki
```

For example:

```
$ hg clone http://bitbucket.org/sarahmaddox/sarahmaddox/wiki
```

or

```
$ git clone http://bitbucket.org/alui/alui-git/wiki
```

Wiki pages are normal files, with the `.wiki extension`. You can edit them locally, as well as creating new ones.

For more about cloning a repository and pushing updates back to bitbucket, see Sharing Code in bitbucket.

Detailed Wiki How To Guides

- Using Syntax Highlighting in a Wiki
- Adding Images to a Wiki Page
- Adding a Table of Contents to a Wiki
- Linking to an Issue from a Wiki Page
- Linking to a Changeset from a Wiki Page
- Linking to a File from a Wiki Page
- Using Wiki Markup
- Making a Wiki Private or Public
- Enabling a Wiki

RELATED TOPICS

bitbucket 101
Repository privacy, permissions, and more

Using Syntax Highlighting in a Wiki

You can highlight snippets of text on your bitbucket wiki pages. We use the excellent Pygments library.

Here is an example of some Python code, formatted in the bitbucket wiki:

```
1  def wiki_rocks(text):
2          formatter = lambda t: "funky"+t
3          return formatter(text)
```

Here is the syntax you will use to format the code as shown above:

```
{{{
#!python

def wiki_rocks(text):
 formatter = lambda t: "funky"+t
 return formatter(text)
}}}
```

See the library of Pygment lexers. bitbucket supports the short name or the mimetype of anything in that library.

Adding Images to a Wiki Page

To add an image to a wiki page, you need to clone the wiki repository to your local computer, add the image file to the repository, include it on the wiki page and then push the changes back to bitbucket.

1. Clone the wiki to your local computer:

```
hg clone  http://bitbucket.org/MY_USER/MY_REPO/wiki/
```

or

```
git clone  http://bitbucket.org/MY_USER/MY_REPO/wiki/
```

2. Copy the image to your local clone directory.
3. Commit the image to the local repository:

```
hg commit -A -m "added image"
```

or

```
git commit -A -m "added image"
```

4. Push the new changeset to the wiki on bitbucket:

```
hg push https://bitbucket.org/MY_USER/MY_REPO/wiki/
```

or

```
git push https://bitbucket.org/MY_USER/MY_REPO/wiki/
```

5. Include the image in your wiki page using this syntax:

```
{{image.png|title}}
```

For example:

```
{{image.png|My favourite chocolate picture}}
```

Note: The image will **not** show up in the preview when you edit the page using the bitbucket wiki editor.

RELATED TOPICS

Using Wiki Markup in bitbucket
bitbucket 101

Adding a Table of Contents to a Wiki

Insert `<<toc>>` into the page that needs the table of contents ('ToC'). You can also create a table of contents for a directory.

This is the basic syntax to generate a table of contents:

```
<<toc [file/folder]  [level=3]>>
```

RELATED TOPICS

Using Wiki Markup

Linking to an Issue from a Wiki Page

Use the following markup to display a link to an issue in your bitbucket issue tracker.

- Add a link to the issue with ID '1':

```
<<issue 1>>
```

- Add a link to all issues with status open or new, with link text 'All open issues'.

```
<<query ?status=open&status=new All open issues>>
```

- Add a list of all issues for milestone 'Beta 1', with links and status for each issue

```
<<milestone Beta 1>>
```

- List all issues that match the query (status is open or new) in one of two optional styles, or show the number of issues that match the query:

```
<<issues ?status=open&status=new count|list|compact>>
```

- Show a full list of open or new issues, in the same layout as the milestone macro:

```
<<issues ?status=open&status=new list>>
```

- Show a list of open or new issues, but only the issue ID with a link to each issue in the tracker:

```
<<issues ?status=open&status=new compact>>
```

RELATED TOPICS

Using Wiki Markup in bitbucket

Linking to a Changeset from a Wiki Page

Use the following markup to display a link to the diff view for changeset number 4c1cff6485f5:

```
<<changeset 4c1cff6485f5>>
```

RELATED TOPICS

Using Wiki Markup in bitbucket

Linking to a File from a Wiki Page

Use the following markup to link to a file for the given revision and line number, where both are optional:

```
<<file path/to/file [revision] [linenumber]>>
<<file [rev:]path/to/file[#L123]>>
```

Notes:

- By default, if you do not specify the revision, the link will point to 'tip'.
- By default, if you do not specify the line number, the link will point to line 0.
- This link syntax does not work with tags.

RELATED TOPICS

Using Wiki Markup in bitbucket

Using Wiki Markup

Your bitbucket wiki uses the Creole syntax, and is fully compatible with the Creole 1.0 specification.

References:

- All Creole markup elements.
- Additional markup elements:
  - bitbucket supports the following additional elements:
    - Plug-in/Extension
    - Monospace
    - Superscript
    - Subscript
    - Underline
    - Definition lists
  - bitbucket does *not* support these additional elements:
    - ❌ Alternate link syntax
    - ❌ Indented paragraphs

RELATED TOPICS

Using Syntax Highlighting in a Wiki

Making a Wiki Private or Public

You can set your bitbucket wiki as private or public.

Public versus private: A private repository is visible only to people who are explicitly given permission to see it. A public repository is visible to everyone. The same applies to the bitbucket issue tracker and wiki, which can be either private or public.

Separate settings for repository, wiki and issue tracker: You can set your bitbucket repository, wiki and issue tracker as private or public, independently of each other. For example, you can hide your code from the world by setting your repository as private, but let people see your documentation and issues by marking your wiki and issue tracker as public. Or you could set your repository and wiki as public but keep your issue tracker private. And so on.

Swapping the settings around: It is OK change the settings from private to public, or public to private, at any time.

Permissions

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Configuring the Private/Public Settings

1. Go to the bitbucket **Admin** tab.
2. Tick the **Wiki is private** check box to make your wiki private. Untick it to make the wiki public.
3. Click **Save repository settings**.

Screenshot: Private/Public Settings on the bitbucket Admin tab

RELATED TOPICS

Repository privacy, permissions, and more

Enabling a Wiki

You can choose whether your bitbucket project will include a wiki or not.

Permissions

You need administrator permission for the repository, to have access to the bitbucket **Admin** tab. See Repository privacy, permissions, and more.

Enabling the Wiki

1. Go to the bitbucket repository **Admin** tab.
2. Click **Wiki settings** in the navigation bar on the left side of the screen.
3. Select **Private wiki** or Public wiki depending on your preference.
4. Click the **Wiki** tab to edit your new wiki.

RELATED TOPICS

Making a Wiki Private or Public

REST APIs, writing brokers, and OAuth

Getting Started

There are two main ways to develop with bitbucket – using our remote API or developing a broker.

Main Topics

REST APIs
Use the REST APIs to interact with a bitbucket repository, wiki, issue tracker and with other users.

Brokers
Learn how to write your own broker. bitbucket offers integration with external services by allowing a set of services, or 'brokers', to run at certain events. These brokers are simple Python scripts. For example, there's a broker for sending emails, and one for posting back to an arbitrary URL.

OAuth on bitbucket
bitbucket's OAuth support enables third-party applications to interact with repositories hosted on the site.

Help

... Full table of contents

bitbucket user's guides

Forums

Feature requests and bug reports

Bitbucket
(Unlimited DVCS Code Hosting, Free)
Pull requests across branches
SourceTree 1.3 our FREE DVCS client now available
How Atlassian migrated from SVN to Git
Follow Up On Our Downtime Last Week

Introducing Keyboard Shortcuts

REST API RESOURCES

Changesets

Emails

Events

Followers

Groups

Groups Privileges

Invitations

Issue Comments

Issues

Privileges

Repositories

Services

Source

SSH Keys

Users

Wiki

Developing Python Scripts for bitbucket

- Writing Brokers for bitbucket

Writing Brokers for bitbucket

bitbucket offers integration with external services by allowing a set of services, or 'brokers', to run at certain events.

These brokers are simple Python scripts that receive information about the event and then take action. For example, there is a broker for sending emails, and another for posting the event to a given URL. You can also write your own broker, as described below. You should use Python 2.5 or higher for writing a new broker.

Please take a look at the guide to managing bitbucket services.

Broker Architecture

A broker is a small Python script. It receives a payload in the form of a dictionary containing at least three top-level keys:

- The **repository** key contains information about the repository, including the name, the slug and the absolute URL.
- The **commits** key is a list containing information about each commit in the push, including the author, commit message, files affected and timestamp.
- The **service** key is a dictionary containing service-specific information. For example, for the Twitter broker there will be a username and a password.

Steps in the flow:

1. User initiates a push.
2. bitbucket dispatches a message to 'Oberon', the service integration daemon.
3. Oberon checks if the repository has any services enabled, and if so, constructs a payload and sends it off to the broker.
4. The broker receives and analyses the payload.
5. The broker takes action. For example, it pings a URL, sends an email, or does whatever you design it to do.

More Detail about the Payload

The payload, as summarised above, is a uniform dictionary containing information on the repository, the commits and the service-specific values.

Here is an example payload for the Twitter broker:

```
{'broker': u'twitter',
 'commits': [{ 'author': u'jespern',
               'files': [{'file': u'media/css/layout.css',
                          'type': u'modified'},
                         {'file': u'apps/bb/views.py',
                          'type': u'modified'},
                         {'file': u'templates/issues/issue.html',
                          'type': u'modified'}],
               'message': u'adding bump button, issue #206 fixed',
               'node': u'e71c63bcc05e',
               'revision': 1650,
               'size': 684}],
 'repository': { 'absolute_url': u'/jespern/bitbucket/',
                 'name': u'bitbucket',
                 'owner': u'jespern',
                 'slug': u'bitbucket',
                 'website': u'http://bitbucket.org/'},
 'service': {'password': u'bar', u'username': u'foo'}}
```

This particular payload contains only a single changeset. If more changesets were present, they would look the same and be in the same list.

The service key contains specific information for the service (Twitter, in this case). If your broker defines a field, such as a URL, you will receive a URL argument in the service key. All field names are converted to lower case.

Writing a Broker

We encourage you to contribute brokers, as they are extremely simple to write. For example, this is the POST broker:

```
import urllib
from brokers import BaseBroker
from django.utils import simplejson as sj

class URLOpener(urllib.FancyURLopener):
    version = 'bitbucket.org'

class Post(BaseBroker):
    def handle(self, payload):
        url = payload['service']['url']

        del payload['service']
        del payload['broker']

        post_load = { 'payload': sj.dumps(payload) }

        opener = self.get_local('opener', URLOpener)
        opener.open(url, urllib.urlencode(post_load))
```

That's all there is to it!

All brokers **must** subclass `BaseBroker` from the `brokers` module. This class is a metaclass that keeps track of the registered brokers and their versions, so that we can reload them without restarting Oberon. The class also makes a method available to you, namely `get_local`, which is a thread-safe way of instantiating. This is important for `urllib` for example, as we do not want authentication or caching to be jumbled among the threads.

Your broker **must** implement one method that we will call `handle`. This function will receive a single argument, the payload.

Notice that we also delete the `service` and `broker` keys from our payload, as there is no reason to expose those to the endpoint.

Testing your Broker

We ask you to kindly test your brokers before submitting them to us for certification. An easy way to do this is to use the example payload shown above, and send it off to your broker's `handle` method.

Submitting your Broker

Once you have written your broker and you are satisfied with it, send it off to `support@bitbucket.org`, or advertise it on the mailing list, or come on IRC! See how to contact us.

We ask that you license your contributions using one of the following licenses:

- Apache
- BSD
- Eclipse
- MIT
- Mozilla

Using the bitbucket REST APIs

You can use bitbucket's REST APIs to interact programmatically with a bitbucket repository, wiki, issue tracker and with other users.

> **ℹ The APIs are experimental and beta**
>
> bitbucket's REST APIs are experimental at this stage. We will be adding to them in future releases. So please expect some API changes. 😊 We will be delighted if you use the APIs and give us your feedback via our issue tracker.

Introduction to bitbucket's REST APIs

bitbucket's REST APIs provide access to resources (data entities) via URI paths. To use a REST API, your application will make an HTTP request and parse the response. By default, the response format is JSON. If you wish, you can request XML or YAML instead of JSON. Your methods will be the standard HTTP methods like GET, PUT, POST and DELETE.

Because the REST API is based on open standards, you can use any web development language to access the API.

bitbucket's APIs provide the following REST resources:

- Changesets
- Emails
- Events
- Followers
- Groups
- Groups Privileges
- Invitations
- Issue Comments
- Issues
- Privileges
- Repositories
- Services
- Source
- SSH Keys
- Users
- Wiki

Access the API via SSL

The bitbucket API works only over HTTPS.

Output Formats

By default, our API will speak JSON. You can use the '`?format=`' query string parameter to override this behaviour. JSON, XML and YAML emitters are available. For example, use following query string:

```
?format=yaml
```

Time zones

Any timestamp returned whose key contains "utc" is in Coordinated Universal Time. All other timestamps are in the local time of Amsterdam. Amsterdam timestamps are now deprecated. If you're using them in your application, please switch to the UTC equivalents.

bitbucket REST Resources

Changesets

The bitbucket 'changesets' REST resource provides functionality for getting information about changesets in your bitbucket repository.

Overview

This is a read-only resource.

Example, getting a list of changesets:

```
$ curl https://api.bitbucket.org/1.0/repositories/jespern/django-piston/changesets/

{
    "count": 219,
    "start": "tip",
    "limit": 15,
    "changesets": [
        {
            "node": "21a24da68710",
            "files": [
                {
                    "type": "modified",
                    "file": "piston/oauth.py"
                }
            ],
            "author": "jespern",
            "timestamp": "2009-09-08 12:49:43",
            "branch": "default",
            "message": "oauth 1.0a spec ready oauth.py",
            "revision": 204,
            "size": 4166
        },
    ...
}
```

Another example, getting a specific changeset:

```
$ curl
https://api.bitbucket.org/1.0/repositories/jespern/django-piston/changesets/fa57572a9acf/

{
    "author": "jespern",
    "branch": "default",
    "files": [
        {
            "file": "piston/handler.py",
            "type": "modified"
        }
    ],
    "message": "warning in case of a model being registered twice under different handlers,
configurable via PISTON_IGNORE_DUPE_MODELS",
    "node": "fa57572a9acf",
    "revision": 207,
    "size": 561,
    "timestamp": "2009-09-09 12:55:07"
}
```

To get more information about the files that are affected by the changeset:

```
$ curl
https://api.bitbucket.org/1.0/repositories/jespern/django-piston/changesets/c9fe89f3ec79/diffstat/

[
    {
        "type": "modified",
        "file": "piston/emitters.py",
        "diffstat": {
            "removed": 2,
            "added": 2
        }
    },
    {
        "type": "modified",
        "file": "piston/resource.py",
        "diffstat": {
            "removed": 10,
            "added": 6
        }
    }
]
```

Getting a List of Changesets

```
GET /repositories/USERNAME/REPO_SLUG/changesets/NODE/
```

Returns a list of changesets for a repository.

Parameters:

- USERNAME: The owner username.
- REPO_SLUG: The slug of the repository.
- NODE: The specific hash to lookup (excludes start and limit). The default is no node.

In addition, you can add the following query string parameters:

- start: The node to begin with. The default is 'tip'.
- limit: The number of changesets returned. The default is 15.

Example with a query string parameter:

```
$ curl https://api.bitbucket.org/1.0/repositories/jespern/django-piston/changesets/?limit=2
```

RELATED TOPICS

Using the bitbucket REST APIs

Emails

The bitbucket 'emails' REST endpoint provides functionality for adding and removing email addresses on your account. It's the API counterpart of the emails widget on the account page.

Note that this API requires authentication.

Querying Your Addresses

```
$ curl --user evzijst:password https://api.bitbucket.org/1.0/emails/
[
    {
        "active": true,
        "email": "evzijst@atlassian.com",
        "primary": true
    },
    {
        "active": true,
        "email": "erik@atlassian.com",
        "primary": false
    }
]
```

The command above list all addresses you have on your account.

You can also request the status of a single one of your addresses (mind the trailing slash):

```
$ curl --user evzijst:password https://api.bitbucket.org/1.0/emails/erik@atlassian.com/
{
    "active": true,
    "email": "erik@atlassian.com",
    "primary": false
}
```

Status Codes

200: Ok

On normal successful execution.

401: Unauthorized

Returned if you haven't provided your login credentials.

Adding Email Addresses

To add a new address, we issue a PUT:

```
$ curl --request PUT --user evzijst:password
https://api.bitbucket.org/1.0/emails/erik.van.zijst@atlassian.com/
{
    "active": false,
    "email": "erik.van.zijst@atlassian.com",
    "primary": false
}
```

Note that this address gets flagged as "inactive" until you have confirmed ownership. While it is flagged as inactive, the address will not be

used by bitbucket. Hence, a changeset on a repo whose username is `erik.van.zijst@atlassian.com` will not get linked to my account yet.

When you add an address, bitbucket sends an email to the new address that contains a link to activate it. After you've clicked that link, the address will become "active" and available for use on the site. Changesets will automatically get linked to your account.

When you run the same command again, bitbucket will send a new confirmation email that has a new, unique activation link in it which supersedes the link in the first email. Note that this is how the "resend confirmation" link on the account page works.

> ℹ️ Note that it might take some time for all relevant pages to reflect the email changes, due to caching.
>
> Also note that depending on the client you use, you might get a "411 Length Required" response. If this happens, your client is not sending the `Content-Length: 0` header. You might need to add this yourself:
>
> ```
> curl --request PUT --header "Content-Length: 0" --user user:pass
> https://api.bitbucket.org/1.0/emails/erik.van.zijst@atlassian.com/
> ```

Setting Email Address As Primary

Setting a confirmed email address is almost the same as adding a new email address, except we send `primary` data along with the request:

```
$ curl --request POST --user evzijst:password
https://api.bitbucket.org/1.0/emails/erik@atlassian.com/ --data "primary=true"
{
    "active": true,
    "email": "erik@atlassian.com",
    "primary": true
}
```

Note that an account can only have one primary email address, so `evzijst@atlassian.com` becomes non-primary:

```
$ curl --user evzijst:password https://api.bitbucket.org/1.0/emails/
[
    {
        "active": true,
        "email": "evzijst@atlassian.com",
        "primary": false
    },
    {
        "active": true,
        "email": "erik@atlassian.com",
        "primary": true
    },
    {
        "active": false,
        "email": "erik.van.zijst@atlassian.com",
        "primary": false
    }
]
```

Primary email address is the main contact address on the account.

More information: Setting Your Email Preferences

Status Codes

201: Created

On normal successful execution.

400: Bad Request

Returned if the email address is not syntactically valid.

401: Unauthorized

Returned if you haven't provided your login credentials.

403: Forbidden

Returned if the specified address is already in use by another user (two users cannot share the same address).

Deleting Email Addresses

To delete an address, run the following command:

```
$ curl --request DELETE --user evzijst:password
https://api.bitbucket.org/1.0/emails/erik.van.zijst@atlassian.com/
```

Status Codes

204: No Content

On normal successful execution.

401: Unauthorized

Returned if you have not provided your login credentials.

Also returned if you the address you are trying to delete is your primary email address.

404: Not Found

Returned if the address you are trying to delete isn't part of your account

409: Conflict

Returned if the address you are trying to delete is the last email address on your account

Events

The bitbucket `'events'` REST resource provides information about user events and repository events.

Getting a List of Events for a User

```
GET /users/USERNAME/events/
```

Parameters:

- `USERNAME`: The user's username.

In addition, you can add the following query string parameters:

- `start`: The offset to start with. The default is '0'.
- `limit`: The number of events returned. The default is 25.

Example with a query string parameter:

```
$ curl https://api.bitbucket.org/1.0/users/jespern/events/?limit=5
```

Getting a List of Events for a Repository

```
GET /repositories/USERNAME/REPO_SLUG/events/
```

Parameters:

- `USERNAME`: The owner's username.
- `REPO_SLUG`: The slug of the repository.

In addition, you can add the following query string parameters:

- `start`: The offset to start with. The default is '0'.

- `limit`: The number of events returned. The default is 25.
- `type`: A filter for the event type to retrieve. e.g. `issue_comment`

Example with a query string parameter:

```
$ curl https://api.bitbucket.org/1.0/repositories/sarahmaddox/sarahmaddox/events/?limit=5
```

RELATED TOPICS

[Using the bitbucket REST APIs](#)

Followers

The bitbucket '`followers`' REST resource provides functionality to list the followers of a user, repository or issue.

Overview

In bitbucket, people can follow users, repositories and issues. Since these lists can be quite big, the default UI does not list the followers individually. It just displays the number of followers. You can use the REST APIs to retrieve a list of followers.

Getting the List of Followers of a User

```
GET /users/USERNAME/followers/
```

Parameters:

- `USERNAME`: The username.

Getting the List of Followers of a Repository

```
GET /repositories/USERNAME/REPO_SLUG/followers/
```

Parameters:

- `USERNAME`: The owner's username.
- `REPO_SLUG`: The slug of the repository.

Getting the List of Followers of an Issue

```
GET /repositories/USERNAME/REPO_SLUG/issues/ISSUE_ID/followers/
```

Parameters:

- `USERNAME`: The owner's username.
- `REPO_SLUG`: The slug of the repository.
- `ISSUE_ID`: The ID of the issue.

Getting the List of Repositories you follow

```
GET /user/follows/
```

This will return the list of repositories the authenticated user follows.

RELATED TOPICS

[Using the bitbucket REST APIs](#)

Groups

The bitbucket '`groups`' REST resource provides functionality for querying information about groups, creating new ones, updating memberships, and deleting them.

Querying Your Groups

```
$ curl --request GET --user username:password https://api.bitbucket.org/1.0/groups/username/
[
    {
        "name": "developers",
        "permission": "read",
        "auto_add": false,
        "members": [
            {
                "username": "jstepka",
                "first_name": "Justen",
                "last_name": "Stepka",
                "avatar":
"https://secure.gravatar.com/avatar/12e5043280f67465b68ac42985082498?d=identicon&s=32",
                "resource_uri": "/1.0/users/jstepka"
            },
            {
                "username": "detkin",
                "first_name": "Dylan",
                "last_name": "Etkin",
                "avatar":
"https://secure.gravatar.com/avatar/e1ef8ef737e394a17ffbc27c889c2b22?d=identicon&s=32",
                "resource_uri": "/1.0/users/detkin"
            }
        ],
        "owner": {
            "username": "baratrion",
            "first_name": "Mehmet S",
            "last_name": "Catalbas",
            "avatar":
"https://secure.gravatar.com/avatar/55a1369161d3a648729b59cabf160e70?d=identicon&s=32",
            "resource_uri": "/1.0/users/baratrion"
        },
        "slug": "developers"
    }
]
```

The command above lists all the groups for the account owner.

While other fields are descriptive `auto_add`, `permission`, and `slug` fields need clarification. `auto_add` is used in conjunction with `permission` to enable default access to newly created repositories. `slug` is used to fetch group specific information in API calls.

Alternatively, you can use a validated email address in place of a username for the Groups API:

```
$ curl --request GET --user username:password
https://api.bitbucket.org/1.0/groups/username@example.com/
[
    {
        "name": "developers",
        "permission": "read",
        "auto_add": false,
        "members": [
            {
                "username": "jstepka",
                "first_name": "Justen",
                "last_name": "Stepka",
                "avatar":
"https://secure.gravatar.com/avatar/12e5043280f67465b68ac42985082498?d=identicon&s=32",
                "resource_uri": "/1.0/users/jstepka"
            },
            {
                "username": "detkin",
                "first_name": "Dylan",
                "last_name": "Etkin",
                "avatar":
"https://secure.gravatar.com/avatar/e1ef8ef737e394a17ffbc27c889c2b22?d=identicon&s=32",
                "resource_uri": "/1.0/users/detkin"
            }
        ],
        "owner": {
            "username": "baratrion",
            "first_name": "Mehmet S",
            "last_name": "Catalbas",
            "avatar":
"https://secure.gravatar.com/avatar/55a1369161d3a648729b59cabf160e70?d=identicon&s=32",
            "resource_uri": "/1.0/users/baratrion"
        },
        "slug": "developers"
    }
]
```

Adding Groups

To create a group called `designers`:

```
$ curl --request POST --user username:password
https://api.bitbucket.org/1.0/groups/username@example.com/ --data "name=designers"
{
    "name": "designers",
    "permission": "read",
    "auto_add": false,
    "members": [],
    "owner": {
        "username": "baratrion",
        "first_name": "Mehmet S",
        "last_name": "Catalbas",
        "avatar":
"https://secure.gravatar.com/avatar/55a1369161d3a648729b59cabf160e70?d=identicon&s=32",
        "resource_uri": "/1.0/users/baratrion"
    },
    "slug": "designers"
}
```

This will return 200 status code on success with the response body containing details about the newly created group.

Updating Groups

To update the settings of a group, issue a `PUT` request with your username and the group slug.

Let's change the name of the `designers` group to `developers` and grant default write access to it for all newly created repositories.

```
$ curl --request PUT --user username:password
https://api.bitbucket.org/1.0/groups/username@example.com/designers/ --data
"name=developers&permission=write&auto_add=true"
{
    "name": "developers",
    "permission": "write",
    "auto_add": true,
    "members": [],
    "owner": {
        "username": "baratrion",
        "first_name": "Mehmet S",
        "last_name": "Catalbas",
        "avatar":
"https://secure.gravatar.com/avatar/55a1369161d3a648729b59cabf160e70?d=identicon&s=32",
        "resource_uri": "/1.0/users/baratrion"
    },
    "slug": "developers"
}
```

You may need to add `--header "Content-Length: 0"` when making `PUT` requests.

Deleting Groups

To delete a group:

```
$ curl --request DELETE --user username:password
https://api.bitbucket.org/1.0/groups/username/test/
```

Listing Members of Groups

To list all the members of the `developers` group:

```
$ curl --request GET --user username:password
https://api.bitbucket.org/1.0/groups/username/developers/members/
[
    {
        "username": "jstepka",
        "first_name": "Justen",
        "last_name": "Stepka",
        "avatar":
"https://secure.gravatar.com/avatar/12e5043280f67465b68ac42985082498?d=identicon&s=32",
        "resource_uri": "/1.0/users/jstepka"
    },
    {
        "username": "detkin",
        "first_name": "Dylan",
        "last_name": "Etkin",
        "avatar":
"https://secure.gravatar.com/avatar/e1ef8ef737e394a17ffbc27c889c2b22?d=identicon&s=32",
        "resource_uri": "/1.0/users/detkin"
    }
]
```

Adding Members to Groups

To add a member with the username `brao` to the group `developers`:

```
curl --request PUT --user username:password
https://api.bitbucket.org/1.0/groups/username/developers/members/brao/
```

Alternatively, you can add the member `brao` to a group using one of his validated email address.

Removing Members from Groups

To remove a member with the username `brao` from the group `developers`:

```
$ curl --request DELETE --user username:password
https://api.bitbucket.org/1.0/groups/username/developers/members/brao/
```

Alternatively, you can remove the user members `brao` from a group using one of his validated email address.

Status Codes

401: Unauthorized

Returned if you try to access somebody else's group.

400: Bad Request

- Returned when the permission is not `read`, `write`, or `admin`.

- Returned when trying to add yourself to one of your groups.

404: Not Found

Returned if the specified resource does not exist.

409: Conflict/Duplicate

Returned when trying to add a member and if member is already in a group.

Groups Privileges

The bitbucket `'groups privileges'` REST resource provides functionality for querying and manipulating the group privileges (permissions) of your repositories.

Querying for Privileges

```
$ curl --user mcatalbas:password -X GET
https://api.bitbucket.org/1.0/group-privileges/mcatalbas/test/
[
    {
        "repo": "mcatalbas/test",
        "privilege": "write",
        "group": {
            "owner": {
                "username": "mcatalbas",
                "first_name": "Mehmet",
                "last_name": "Catalbas"
            },
            "name": "developers",
            "members": [
                {
                    "username": "nvenegas",
                    "first_name": "Nicolas",
                    "last_name": "Venegas"
                },
                {
                    "username": "brao",
                    "first_name": "Brodie",
                    "last_name": "Rao"
                }
            ],
            "slug": "developers"
        },
        "repository": {
            "owner": {
                "username": "mcatalbas",
                "first_name": "Mehmet",
                "last_name": "Catalbas"
            },
            "name": "test",
            "slug": "test"
        }
    },
    {
        "repo": "mcatalbas/test",
        "privilege": "admin",
        "group": {
            "owner": {
                "username": "mcatalbas",
                "first_name": "Mehmet",
                "last_name": "Catalbas"
            },
            "name": "managers",
            "members": [
                {
                    "username": "detkin",
                    "first_name": "Dylan",
                    "last_name": "Etkin"
                },
                {
                    "username": "jnoehr",
                    "first_name": "Jesper",
                    "last_name": "Noehr"
                }
            ],
            "slug": "managers"
        },
        "repository": {
            "owner": {
                "username": "mcatalbas",
                "first_name": "Mehmet",
                "last_name": "Catalbas"
            },
            "name": "test",
            "slug": "test"
        }
    }
]
```

The command above request a list of all privileges for the `mcatalbas/test` repository. The result shows that the group `developers` has write access and the group `managers` has admin access to the repository.

The reason for having both `repo` and `repository` objects is to remain consistent with the Privileges REST API. `repo` will most likely to be removed in the future, and `repository` will be the only object representing a repository.

You can also use other account owner's groups that you are a member of when querying and manipulating privileges of your own repositories. For example, to query `mcatalbas`'s repository `bugfix-331`, for the account owner's `atlassian` group `staff` memberships details:

```
$ curl --request GET --user mcatalbas:password
https://api.bitbucket.org/1.0/group-privileges/mcatalbas/bugfix-331/atlassian/developers
[
    {
        "repo": "mcatalbas/bugfix-331",
        "privilege": "write",
        "group": {
            "owner": {
                "username": "atlassian",
                "first_name": "",
                "last_name": ""
            },
            "name": "staff",
            "members": [
                {
                    "username": "nvenegas",
                    "first_name": "Nicolas",
                    "last_name": "Venegas"
                },
                {
                    "username": "brao",
                    "first_name": "Brodie",
                    "last_name": "Rao"
                }
            ],
            "slug": "staff"
        },
        "repository": {
            "owner": {
                "username": "mcatalbas",
                "first_name": "Mehmet",
                "last_name": "Catalbas"
            },
            "name": "bugfix-331",
            "slug": "bugfix-331"
        }
    }
]
```

List All Privileges

```
$ curl --request GET --user mcatalbas:password
https://api.bitbucket.org/1.0/group-privileges/mcatalbas/
```

Filters

You can use the `filter=read|write|admin` query parameter to limit your results to a specific privilege level:

```
$ curl --request GET --user mcatalbas:password
https://api.bitbucket.org/1.0/group-privileges/mcatalbas/?filter=admin
```

The `private=true` query parameter can be used to only include private repositories:

```
$ curl --request GET --user mcatalbas:password
https://api.bitbucket.org/1.0/group-privileges/mcatalbas/?private=true
```

Granting Privileges

To grant the group `mcatalbas/sys-admins` read only access to the repository `mcatalbas/test`:

```
$ curl --request PUT --user mcatalbas:password
https://api.bitbucket.org/1.0/group-privileges/mcatalbas/test/mcatalbas/sys-admins --data read
```

Status Codes

401: Unauthorized

Returned if the requester does not have "admin" privileges for the repository.

403: Forbidden

Returned if the repository owner is at or over the user limit when performing a PUT operation, e.g. granting privileges to a group.

404: Not Found

Returned if the specified group does not exist.

Revoking Privileges

To revoke privileges of the group `developers` (of `mcatalbas`') from the repository `test` (of `mcatalbas`'):

```
$ curl --request DELETE --user mcatalbas:password
https://api.bitbucket.org/1.0/group-privileges/mcatalbas/test/mcatalbas/developers
```

To revoke privileges of the group `developers` from all of your repositories:

```
$ curl --request DELETE --user mcatalbas:password
https://api.bitbucket.org/1.0/group-privileges/mcatalbas/mcatalbas/developers
```

All deletes return a 200 status code and an empty response body on success.

RELATED TOPICS

Using the bitbucket REST APIs

Invitations

The bitbucket `'invitations'` REST resource allows repository administrators to send email invitations to grant read, write, or admin privileges to a repository.

Once the invitation receipt URL, provided in the email, is visited, that person is automatically granted the privileges specified by the invitation.

Sending an invitation

To send an invitation to grant write privileges to `john@example.com` for the repository `roger/ramjet`:

```
$ curl --user roger:password --request POST --data permission=write \
       'https://api.bitbucket.org/1.0/invitations/roger/ramjet/john@example.com'
```

This will return a 200 status code on success with the response body containing details about the invitation:

```
{
    "sent_on": "2011-01-11 01:03:31",
    "permission": "write",
    "invited_by": {
        "username": "roger",
        "first_name": "Roger",
        "last_name": "Ramjet",
        "avatar":
"https://secure.gravatar.com/avatar/55a1369161d3a648729b59cabf160e70?d=identicon&s=32",
        "resource_uri": "/1.0/users/roger/"
    },
    "repository": {
        "website": "",
        "read_only": false,
        "has_wiki": true,
        "last_updated": "2010-12-24 05:26:20",
        "name": "ramjet",
        "language": "",
        "deleted": false,
        "is_mq": false,
        "mq_of": null,
        "created_on": "2010-12-24 05:26:20",
        "fork_of": null,
        "email_writers": true,
        "size": 4096,
        "owner": "roger",
        "has_issues": true,
        "no_public_forks": false,
        "email_mailinglist": "",
        "is_fork": false,
        "slug": "ramjet",
        "is_private": true,
        "description": ""
    },
    "email": "john@example.com"
}
```

Valid values for `permission` are:

- `read` (except for public repositories)
- `write`
- `admin`

Status Codes

200 OK

Returned when the invitation was successfully created and queued for sending. The response body will contain details of the invitation.

400 Bad Request

Returned when

- the repository (username and repository name combination, e.g., `roger/ramjet`) is not provided

- the email address is invalid

- the permission is not one of `read`, `write`, or `admin`, or `read` is supplied for a public repository

401 Unauthorized

Returned when you don't provide valid log in credentials when requesting the resource or you are not an administrator for the repository.

404 Not Found

Returned when the repository does not exist.

409 Conflict

Returned when the provided email address has already been invited to the repository.

Issue Comments

The 'issue comments' REST resource allows you to create, read, update, and delete bitbucket issue tracker comments. When you create, update, or delete an issue comment you must provide a username/password combination. To update and delete, you must either be the comment author or a user with admin permissions on the repo.

Overview

To list all the comments associated with issue 3210 on repository site/master:

```
$ curl https://api.bitbucket.org/1.0/repositories/site/master/issues/3210/comments/

[
    {
        "content": "I second this!\n\nAm going to implement the same kludge as
Arothian..\n\nWas surprised to see that it wasn't implemented already.\n\nI am glad that you've
got an API, of course. :)",
        "author_info": {
            "username": "jacmoe",
            "first_name": "Jacob",
            "last_name": "Moen",
            "avatar":
"https://secure.gravatar.com/avatar/cddd845778affc94b6f5913c57d1e96b?d=identicon&s=32",
            "resource_uri": "/1.0/users/jacmoe"
        },
        "comment_id": 834548,
        "utc_updated_on": "2011-12-03 04:29:49+00:00",
        "utc_created_on": "2011-12-03 04:28:56+00:00",
        "is_spam": false
    },
    {
        "content": "Hi arothian,\r\n\r\nThats a good suggestion, I will add it to the backlog
of issues around the REST API.\r\n\r\nCheers,\r\n\r\nDylan",
        "author_info": {
            "username": "detkin",
            "first_name": "Dylan",
            "last_name": "Etkin",
            "avatar":
"https://secure.gravatar.com/avatar/e1ef8ef737e394a17ffbc27c889c2b22?d=identicon&s=32",
            "resource_uri": "/1.0/users/detkin"
        },
        "comment_id": 743643,
        "utc_updated_on": "2011-10-27 18:20:21+00:00",
        "utc_created_on": "2011-10-27 18:20:21+00:00",
        "is_spam": false
    },
    {
        "content": null,
        "author_info": {
            "username": "detkin",
            "first_name": "Dylan",
            "last_name": "Etkin",
            "avatar":
"https://secure.gravatar.com/avatar/e1ef8ef737e394a17ffbc27c889c2b22?d=identicon&s=32",
            "resource_uri": "/1.0/users/detkin"
        },
        "comment_id": 743642,
        "utc_updated_on": "2011-10-27 18:19:50+00:00",
        "utc_created_on": "2011-10-27 18:19:50+00:00",
        "is_spam": false
    }
]
```

Getting an individual comment

If you specify the comment ID, you can get information on a single comment:

```
$ curl https://api.bitbucket.org/1.0/repositories/site/master/issues/3210/comments/743642/

{
    "content": null,
    "author_info": {
        "username": "detkin",
        "first_name": "Dylan",
        "last_name": "Etkin",
        "avatar":
"https://secure.gravatar.com/avatar/e1ef8ef737e394a17ffbc27c889c2b22?d=identicon&s=32",
        "resource_uri": "/1.0/users/detkin"
    },
    "comment_id": 743642,
    "utc_updated_on": "2011-10-27 18:19:50+00:00",
    "utc_created_on": "2011-10-27 18:19:50+00:00",
    "is_spam": false
}
```

Adding a comment

To add a new comment, issue a `POST` request. The username you authenticate with becomes the comment author. The system populates the other info fields based on the username.

```
$ curl --request POST --user baratrion:password
https://api.bitbucket.org/1.0/repositories/baratrion/test/issues/4/comments/ --data
"content=Hello World"
{
    "content": "Hello World",
    "author_info": {
        "username": "baratrion",
        "first_name": "Mehmet",
        "last_name": "Catalbas",
        "avatar":
"https://secure.gravatar.com/avatar/55a1369161d3a648729b59cabf160e70?d=identicon&s=32",
        "resource_uri": "/1.0/users/baratrion"
    },
    "comment_id": 893636,
    "utc_updated_on": "2011-12-21 01:02:43+00:00",
    "utc_created_on": "2011-12-21 01:02:43+00:00",
    "is_spam": false
}
```

Editing (updating) an existing comment

To edit a comment, issue a `PUT` request:

```
$ curl --request PUT --user baratrion:password
https://api.bitbucket.org/1.0/repositories/baratrion/test/issues/4/comments/893636/ --data
"content=Hello"
{
    "content": "Hello",
    "author_info": {
        "username": "baratrion",
        "first_name": "Mehmet",
        "last_name": "Catalbas",
        "avatar":
"https://secure.gravatar.com/avatar/55a1369161d3a648729b59cabf160e70?d=identicon&s=32",
        "resource_uri": "/1.0/users/baratrion"
    },
    "comment_id": 893636,
    "utc_updated_on": "2011-12-21 01:05:35+00:00",
    "utc_created_on": "2011-12-21 01:02:43+00:00",
    "is_spam": false
}
```

If you have admin permissions on a repository, you can edit any comment on the repository's issue tracker. Otherwise, you can only edit your

own comments.

Deleting a comment

To delete a comment, issue a `DELETE` request:

```
curl --request DELETE --user baratrion:password
https://api.bitbucket.org/1.0/repositories/baratrion/test/issues/4/comments/893636/
```

If you have admin permissions on a repository, you can delete any comment on the repository's issue tracker. Otherwise, you can only delete your own comments.

Issues

The bitbucket `'issues'` REST resource provides functionality for getting information on issues in the bitbucket issue tracker, creating new issues, updating them and deleting them.

Overview

You can access public issues without authentication, but you will only receive a subset of information, and you can't gain access to private repositories' issues. By authenticating, you will get a more detailed set of information, the ability to create issues, as well as access to updating data or deleting issues you have access to.

Example:

```
$ curl https://api.bitbucket.org/1.0/repositories/site/master/issues/
{
    "count": 2131,
    "filter": {},
    "search": null,
    "issues": [
        {
            "status": "resolved",
            "title": "No Fantom programming language",
            "reported_by": {
                "username": "ystrot",
                "first_name": "Yuri",
                "last_name": "Strot",
                "avatar":
"https://bitbucket-assetroot.s3.amazonaws.com:443/c/photos/2011/Jan/18/ystrot-avatar-3606457333-0_a

                "resource_uri": "/1.0/users/ystrot"
            },
            "responsible": {
                "username": "evzijst",
                "first_name": "Erik",
                "last_name": "van Zijst",
                "avatar":
"https://secure.gravatar.com/avatar/3d3f7d1bc4c6386815e2c1e16c2d1c46?d=identicon&s=32",
                "resource_uri": "/1.0/users/evzijst"
            },
            "comment_count": 3,
            "content": "There are more than 30 projects written in Fantom programming language
(http://fantom.org/): https://bitbucket.org/repo/all?name=fantom\r\n\r\nHowever Fantom is not
available in the language list. Could you please add it?",
            "created_on": "2011-05-31 13:30:42",
            "local_id": 2776,
            "follower_count": 1,
            "metadata": {
                "kind": "enhancement",
                "version": null,
                "component": "repository",
                "milestone": null
            },
            "resource_uri": "/1.0/repositories/site/master/issues/2776",
            "is_spam": false
        },
        # more...
    ]
}
```

Getting an Individual Issue

If you specify the issue ID, you can get information on a single issue:

```
$ curl https://api.bitbucket.org/1.0/repositories/jespern/bitbucket/issues/64/
{
    "status": "invalid",
    "title": "easy_install functionality",
    "reported_by": {
        "username": "jespern",
        "first_name": "Jesper",
        "last_name": "Noehr",
        "avatar":
"https://secure.gravatar.com/avatar/b658715b9635ef057daf2a22d4a8f36e?d=identicon&s=32",
        "resource_uri": "/1.0/users/jespern"
    },
    "responsible": {
        "username": "jespern",
        "first_name": "Jesper",
        "last_name": "Noehr",
        "avatar":
"https://secure.gravatar.com/avatar/b658715b9635ef057daf2a22d4a8f36e?d=identicon&s=32",
        "resource_uri": "/1.0/users/jespern"
    },
    "comment_count": 1,
    "content": "Or something like it..?",
    "created_on": "2008-07-21 16:04:35",
    "local_id": 64,
    "follower_count": 3,
    "metadata": {
        "kind": "bug",
        "version": null,
        "component": null,
        "milestone": null
    },
    "resource_uri": "/1.0/repositories/site/master/issues/64",
    "is_spam": false
}
```

Searching

If you do not specify an issue ID, and add the 'search' query string parameter, you can search through issues.

For example, searching for easy_install:

https://secure.gravatar.com/avatar/b658715b9635ef057daf2a22d4a8f36e

```
$ curl "https://api.bitbucket.org/1.0/repositories/site/master/issues/?search=easy_install"
{
    "count": 3,
    "filter": {},
    "search": "easy_install",
    "issues": [
        {
            "status": "new",
            "title": "Getting an Individual Issue via API (BB-1085)",
            "reported_by": {
                "username": "mennanov",
                "first_name": "Renat",
                "last_name": "Mennanov",
                "avatar":
"https://bitbucket-assetroot.s3.amazonaws.com:443/c/photos/2010/Nov/09/1289284926_user_avatar.png",

                "resource_uri": "/1.0/users/mennanov"
            },
            "comment_count": 1,
            "content": "Getting an Individual Issue is useless since it does not provide enough
information about the issue.\r\n\r\n{{\r\n#!bash\r\n\r\n$ curl
https://api.bitbucket.org/1.0/repositories/jespern/bitbucket/issues/64/\r\n\r\n{\r\n
\"title\": \"easy_install functionality\", \r\n  \"comment_count\": 1, \r\n  \"content\": \"Or
something like it..?\", \r\n  \"created_on\": \"2008-07-21 16:04:35\", \r\n  \"reported_by\":
{\r\n    \"username:\" \"madssj\",\r\n     \"resource_uri\": \"/1.0/users/madssj/\"\r\n  },\r\n
\"local_id\": 64, \r\n  \"follower_count\": 2, \r\n  \"metadata\": {\r\n     \"kind\":
\"bug\"\r\n  }, \r\n  \"resource_uri\": \"/1.0/repositories/jespern/bitbucket/issues/64/\",
\r\n  \"is_spam\": false\r\n}\r\n\r\n}}\r\n\r\nIt would be great to get issue's comments, to
whom it is assigned, links to attached files etc..",
            "created_on": "2010-12-04 09:14:25",
            "local_id": 2317,
            "follower_count": 2,
            "metadata": {
                "kind": "enhancement",
                "version": null,
                "component": null,
                "milestone": null
            },
            "resource_uri": "/1.0/repositories/site/master/issues/2317",
            "is_spam": false
        },
        # more...
    ]
}
```

Filtering

You can also apply filter criteria as parameters to narrow the return.

Filter Definition	Operator
contains	~
doesn't contain	!~
begins with	^
ends with	$
is not	!
is	[BBDEV:Nothing]

Parameter	Operators	Notes
title	All	
content	All	Logical Name: Description
version	is & is not ONLY	
milestone	is & is not ONLY	
component	is & is not ONLY	
kind	is & is not ONLY	Logical Name: Type
status	is & is not ONLY	
responsible	is & is not ONLY	
reported_by	All	

When constructing your parameter string, use one of the operators above in combination with the filter as your value. The end result should look like the below example. The parameter (key) and value pair with the desired filter operator preceding the value.

Multiple instances of the same parameter will be treated as **OR** for the overall filter query.

Example

```
$ curl
https://api.bitbucket.org/1.0/repositories/jespern/bitbucket/issues/?title=~easy_install&title=~ren
"count": 2,
    "filter": {
        "title": [
            [
                "~",
                "easy_install"
            ],
            [
                "~",
                "renaming repositories should"
            ]
        ]
    },
    "search": null,
    "issues": [
        {
            "status": "invalid",
            "title": "easy_install functionality",
            "reported_by": {
                "username": "jespern",
                "first_name": "Jesper",
                "last_name": "Noehr",
                "avatar":
"https://secure.gravatar.com/avatar/b658715b9635ef057daf2a22d4a8f36e?d=identicon&s=32",
                "resource_uri": "/1.0/users/jespern"
            },
            "responsible": {
                "username": "jespern",
                "first_name": "Jesper",
                "last_name": "Noehr",
                "avatar":
"https://secure.gravatar.com/avatar/b658715b9635ef057daf2a22d4a8f36e?d=identicon&s=32",
                "resource_uri": "/1.0/users/jespern"
            },
            "comment_count": 1,
            "content": "Or something like it..?",
            "created_on": "2008-07-21 16:04:35",
            "local_id": 64,
            "follower_count": 3,
            "metadata": {
                "kind": "bug",
                "version": null,
                "component": null,
                "milestone": null
            },
            "resource_uri": "/1.0/repositories/site/master/issues/64",
            "is_spam": false
        },
        # more...
    ]
}
```

Specifying an Offset

```
GET /repositories/USERNAME/REPO_SLUG/issues/
```

Path components:

- USERNAME: The owner username.
- REPO_SLUG: The slug of the repository.

Optional query string parameters:

- start: Offset to start at. The default is 0.
- limit: Maximum number of issues returned. The default is 25.

Example with a query string parameter:

```
$ curl https://api.bitbucket.org/1.0/repositories/sarahmaddox/sarahmaddox/issues/?start=25
```

Creating an Issue

```
POST /repositories/USERNAME/REPO_SLUG/issues/
```

Parameters:

- USERNAME: The owner username.
- REPO_SLUG: The slug of the repository.

The following POST data values specify the attributes of the issue:

- title: The title of the new issue.
- content: The content of the new issue.
- component: The component associated with the issue.
- milestone: The milestone associated with the issue.
- version: The version associated with the issue.
- responsible: The username of the person responsible for the issue.
- status: The status of the issue (new, open, resolved, on hold, invalid, duplicate, or wontfix).
- kind: The kind of issue (bug, enhancement, or proposal).

Returns the newly created issue.

Example:

```
$ curl https://api.bitbucket.org/1.0/repositories/sarahmaddox/sarahmaddox/issues/ --data
"title=Issue%20Title&content=Issue%20Content"
```

Updating an issue

```
PUT /repositories/USERNAME/REPO_SLUG/issues/ISSUE_ID/
```

Parameters:

- USERNAME: The owner username.
- REPO_SLUG: The slug of the repository.
- ISSUE_ID: The ID of the issue.

Example:

```
$ curl --request PUT
https://api.bitbucket.org/1.0/repositories/sarahmaddox/sarahmaddox/issues/1/ --data
"content=Updated%20Content"
```

Deleting an Issue

```
DELETE /repositories/USERNAME/REPO_SLUG/issues/ISSUE_ID/
```

Parameters:

- USERNAME: The owner username.
- REPO_SLUG: The slug of the repository.
- ISSUE_ID: The ID of the issue.

Getting Information about Followers of an Issue

See Followers.

RELATED TOPICS

Privileges

The bitbucket 'privileges' REST resource provides functionality for querying and manipulating the privileges (permissions) of your repositories. It allows you to grant specific users access to read, write and or administer your repositories.

Note that repository privileges can only be queried and modified by the repository's owner or by repository administrators.

Querying Privileges

```
$ curl --user evzijst:password https://api.bitbucket.org/1.0/privileges/evzijst/test
[
    {
        "repo": "evzijst/test",
        "privilege": "read",
        "user": {
            "username": "jespern",
            "first_name": "Jesper",
            "last_name": "Noehr"
        }
    },
    {
        "repo": "evzijst/test",
        "privilege": "read",
        "user": {
            "username": "detkin",
            "first_name": "Dylan",
            "last_name": "Etkin"
        }
    },
    {
        "repo": "evzijst/test",
        "privilege": "write",
        "user": {
            "username": "davidchambers",
            "first_name": "David",
            "last_name": "Chambers"
        }
    },
    {
        "repo": "evzijst/test",
        "privilege": "admin",
        "user": {
            "username": "nvenegas",
            "first_name": "Nicolas",
            "last_name": "Venegas"
        }
    }
]
```

The above command requests a list of all privileges on the evzijst/test repository. This is a private repository to which users jespern and detkin have read access, user davidchambers has write access, and nvenegas has full administrative privileges.

Querying can also be done on the individual role level, as well as for all repositories owned by a specific user.
For example, request the access level of user nvenegas on repository evzijst/test:

```
$ curl --user evzijst:password https://api.bitbucket.org/1.0/privileges/evzijst/test/nvenegas
[
    {
        "repo": "evzijst/test",
        "privilege": "admin",
        "user": {
            "username": "nvenegas",
            "first_name": "Nicolas",
            "last_name": "Venegas"
        }
    }
]
```

To list all privileges on all of `evzijst`'s repositories, use `/1.0/privileges/evzijst`

Filters

You can use the `filter=read|write|admin` query parameter to limit your results to a specific role. To list everyone who has write access to my repo, I can run:

```
$ curl --user evzijst:password
https://api.bitbucket.org/1.0/privileges/evzijst/test?filter=write
[
    {
        "repo": "evzijst/test",
        "privilege": "write",
        "user": {
            "username": "davidchambers",
            "first_name": "David",
            "last_name": "Chambers"
        }
    },
    {
        "repo": "evzijst/test",
        "privilege": "admin",
        "user": {
            "username": "nvenegas",
            "first_name": "Nicolas",
            "last_name": "Venegas"
        }
    }
]
```

Note that this result includes administrators, as they too have write access.

Granting Privileges

To grant user `brodie` write access to my repo, I can do a `PUT` on `/1.0/privileges/evzijst/brodie` with the string `write` as request body:

```
$ curl --request PUT --user evzijst:password
https://api.bitbucket.org/1.0/privileges/evzijst/test/brodie --data write
```

This will return a 200 status code on success, with an empty response body.

```
$ curl --request PUT --user evzijst:password
https://api.bitbucket.org/1.0/privileges/evzijst/test/brodie --data read
```

Upgrading or downgrading an existing user's privilege level is also done through this command. When the above command is run, `brodie` will have his "write" access modified to "read". Hence, a user can only have one access level, either "read", "write", or "admin".

Status Codes

401: Unauthorized

Returned if the requester does not have "admin" privileges for the repository.

403: Forbidden

Returned if the repo owner is at or over her account's private user limit when do a PUT (granting privileges to a new user).

404: Not Found

Returned if the specified resource does not exist.

Revoking Privileges

To revoke a user's privileges, simply do a `DELETE`. The following command revokes the privileges of user `brodie`:

```
$ curl --request DELETE --user evzijst:password
https://api.bitbucket.org/1.0/privileges/evzijst/test/brodie
```

To revoke all privileges on repo `evzijst/test` in one go, do:

```
$ curl --request DELETE --user evzijst:password
https://api.bitbucket.org/1.0/privileges/evzijst/test
```

If you want, you can go even further and revoke all privileges on all of your own repositories at once:

```
$ curl --request DELETE --user evzijst:password
https://api.bitbucket.org/1.0/privileges/evzijst
```

All deletes return a 200 status code and empty response body on success.

RELATED TOPICS

Using the bitbucket REST APIs

Repositories

The bitbucket `'repositories'` REST resource provides functionality for getting information about repositories, creating new ones, updating their settings and deleting them.

Overview

You can access public repositories without authentication, but you will only receive a subset of information, and you cannot gain access to private repositories. By authenticating, you will get a more detailed set of information, the ability to create repositories, and the ability to update settings or delete repositories you have access to.

Example:

```
$ curl https://api.bitbucket.org/1.0/users/jespern/

{
  "repositories": [
    {
      "slug": "django-piston",
      "name": "django-piston",
      "resource_uri": "/1.0/repositories/jespern/django-piston/",
      "followers_count": 173,
      "website": "",
      "description": "Piston is a Django mini-framework creating APIs."
    }
  ],
  "user": {
    "username": "jespern",
    "avatar":
"https://secure.gravatar.com/avatar/b658715b9635ef057daf2a22d4a8f36e?d=identicon&s=32",
    "resource_uri": "/1.0/users/jespern/",
    "last_name": "Noehr",
    "first_name": "Jesper"
  }
}
```

Querying the repositories you have access to

If you want to see list of repositories you have access to, you can do:

```
"website": "",
```

```
$ curl --user username:password https://api.bitbucket.org/1.0/user/repositories/
[
    {
        "owner": "jespern",
        "slug": "arriving",
        "is_private": true,
        "name": "arriving"
    },
    {
        "owner": "jespern",
        "slug": "jespern",
        "is_private": true,
        "name": "jespern"
    },
    {
        "owner": "jespern",
        "slug": "jespern-jekyll",
        "is_private": true,
        "name": "jespern-jekyll"
    },
    {
        "owner": "jespern",
        "slug": "bitbucket-issue-query",
        "is_private": true,
        "name": "bitbucket-issue-query"
    },
    {
        "owner": "erik",
        "slug": "org.eclipse.mylyn.bitbucket",
        "is_private": true,
        "name": "org.eclipse.mylyn.bitbucket"
    },
    {
        "owner": "mcatalbas",
        "slug": "codeigniter",
        "is_private": false,
        "name": "CodeIgniter"
    },
    {
        "owner": "brodie",
        "slug": "hg-project",
        "is_private": true,
        "name": "hg-project"
    },
    {
        "owner": "dchambers",
        "slug": "django-piston",
        "is_private": false,
        "name": "django-piston"
    }
]
```

Searching

If you do not specify username and repo, you can search through repositories. You can do that by specifying the 'name' query string parameter.

Example: Searching for 'django-':

```
$ curl "https://api.bitbucket.org/1.0/repositories/?name=django-"

{
  "query": "django-",
  "count": 372,
  "repositories": [
    {
      "website": "",
      "name": "django-registration",
      "description": "A user-registration application for Django.",
      "followers_count": 159,
      "slug": "django-registration",
      "resource_uri": "/1.0/repositories/ubernostrum/django-registration/"
    },
    {
      "website": "",
      "name": "django-profiles",
      "description": "A user-profile application for Django.",
      "followers_count": 60,
      "slug": "django-profiles",
      "resource_uri": "/1.0/repositories/ubernostrum/django-profiles/"
    },
    # many more...
  ]
}
```

Tags and Branches

You can get a list of branches and tags for a given repository.

Get a list of tags for a given repository:

```
GET /repositories/USERNAME/REPO_SLUG/tags/
```

Get a list of branches for a given repository:

```
GET /repositories/USERNAME/REPO_SLUG/branches/
```

Parameters:

- USERNAME: The owner's username.
- REPO_SLUG: The slug of the repository.

Getting a Single Repository

```
GET /repositories/USERNAME/REPO_SLUG/
```

Parameters:

- USERNAME: The owner's username.
- REPO_SLUG: The slug of the repository.

Creating a New Repository

Example:

```
curl --request POST --user username:password https://api.bitbucket.org/1.0/repositories/ --data
name=myrepo --data scm=git
```

You must specify at least the name and the scm parameter in the POST body (x-www-form-urlencoded).

The `scm` parameter can have the value 'hg' or 'git' depending on which type of repository you are trying to create.

Returns the newly created repository on success, otherwise returns a response code other than '200'.
This will create a public repository. To create a private repository, add the `is_private=True` parameter.

Updating an Existing Repository

```
PUT /repositories/USERNAME/REPO_SLUG/
```

Parameters:

- `USERNAME`: The owner's username.
- `REPO_SLUG`: The slug of the repository.

You can send just a single field and new value, and the existing values will be filled in automatically.

Deleting a Repository

```
DELETE /repositories/USERNAME/REPO_SLUG/
```

Parameters:

- `USERNAME`: The owner's username.
- `REPO_SLUG`: The slug of the repository.

Getting Information about Repository Events

See Events.

Getting Information about Followers of a Repository

See Followers.

RELATED TOPICS

Using the bitbucket REST APIs

Services

The bitbucket 'services' REST endpoint provides functionality for adding, removing, and configuring services (brokers) on your repositories.

Note that this API requires authentication.

Querying Services

```
$ curl --user mcatalbas:password
https://api.bitbucket.org/1.0/repositories/mcatalbas/test/services/
[
    {
        "id": 3,
        "service": {
            "fields": [
                {
                    "name": "Email",
                    "value": "mcatalbas@atlassian.com"
                }
            ],
            "type": "Email"
        }
    },
    {
        "id": 4,
        "service": {
            "fields": [
                {
                    "name": "URL",
                    "value": "http://example.com/post"
                }
            ],
            "type": "POST"
        }
    }
]
```

The command above list all services on repository `mcatalbas/test`.

You can also request a single service attached to your repository:

```
$ curl --user mcatalbas:password
https://api.bitbucket.org/1.0/repositories/mcatalbas/test/services/4/
[
    {
        "id": 4,
        "service": {
            "fields": [
                {
                    "name": "URL",
                    "value": "http://example.com/post"
                }
            ],
            "type": "POST"
        }
    }
]
```

Adding Services

To add a new service, issue a `POST` request:

```
$ curl --request POST --user mcatalbas:password
https://api.bitbucket.org/1.0/repositories/mcatalbas/test/services/ --data
"type=post;URL=http://bitbucket.org/post"
{
    "id": 5,
    "service": {
        "fields": [
            {
                "name": "URL",
                "value": "https://bitbucket.org/post"
            }
        ],
        "type": "POST"
    }
}
```

The example above shows how to create a new `POST` service on repository `mcatalbas/test`. Please note that, only the `type` of the service is required in the `POST` data. (Service type is a case-insensitive match.)

All of the service fields that are specific to that service can be supplied, but it is optional. You can update service fields later. (Outlined below.)

Available Services

- type: `POST`
    - fields: `URL`
- type: `FogBugz`
    - fields: `Repository ID`, `CVSSubmit URL`
- type: `Basecamp`
    - fields: `Username`, `Password`, `Discussion URL`
- type: `Lighthouse`
    - fields: `Project ID`, `API Key`, `Subdomain`
- type: `CIA.vc`
    - fields: `Module`, `Project`
- type: `Issues`
    - fields: None
- type: `Email`
    - fields: `Email`
- type: `Email Diff`
    - fields: `Email`
- type: `FriendFeed`
    - fields: `Username`, `Remote Key`, `Format`
- type: `Rietveld`
    - fields: `Email`, `Password`, `URL`
- type: `Superfeedr`
    - fields: None
- type: `Geocommit`
    - fields: None
- type: `Pivotal Tracker`
    - fields: `Token`

> ⚠ Although you can add Twitter service via API, since it requires three-legged OAuth authentication, you still need to configure it via web interface.

Updating Services

To update a service, issue a `PUT` request with the `ID` of the repository service included in the URL:

```
$ curl --request PUT --user mcatalbas:password
https://api.bitbucket.org/1.0/repositories/mcatalbas/test/services/5/ --data
"URL=https://bitbucket.org/new_post"
{
    "id": 5,
    "service": {
        "fields": [
            {
                "name": "URL",
                "value": "https://bitbucket.org/new_post"
            }
        ],
        "type": "POST"
    }
}
```

Deleting Services

To delete a service, issue a `DELETE` request:

```
$ curl --request DELETE --user mcatalbas:password
https://api.bitbucket.org/1.0/repositories/mcatalbas/test/services/5/
```

Status Codes

200: Ok

On normal successful execution.

401: Unauthorized

Returned if you have not provided your login credentials.

Also returned if you are not one of the administrators of the repository.

404: Not Found

Returned if the resource does not exist.

Source

You can use the bitbucket 'src' REST resource to browse directories and view files.

Overview

This is a read-only resource.

Example of browsing a directory:

```
$ curl https://api.bitbucket.org/1.0/repositories/jespern/django-piston/src/tip/piston/

{
    "diretories": [
        "fixtures",
        "templates"
    ],
    "files": [
        {
            "path": "resource.py",
            "revision": "5c2a4cd036ec",
            "size": 8797,
            "timestamp": "2009-09-10 10:28:58"
        },
        {
            "path": "doc.py",
            "revision": "80086e0a8022",
            "size": 5738,
            "timestamp": "2009-09-02 11:23:37"
        },
        # more ...
    ]
}
```

Example of viewing a file:

```
$ curl https://api.bitbucket.org/1.0/repositories/jespern/django-piston/src/tip/piston/utils.py

{
    "data": "import time\nfrom django.http import HttpResponseNotAllowed, ...",
    "node": "7970b070f884",
    "path": "piston/utils.py"
}
```

Instead of getting the file formatted as JSON, you can get the raw file:

```
$ curl https://api.bitbucket.org/1.0/repositories/jespern/django-piston/raw/tip/piston/utils.py
import time
from django.http import HttpResponseNotAllowed, HttpResponseForbidden, HttpResponse,
HttpResponseBadRequest
from django.core....
```

Viewing Directories and Files

```
GET /repositories/USERNAME/REPO_SLUG/REVISION/PATH/
```

Reads either directories or files.

Parameters:

- `USERNAME`: The owner username.
- `REPO_SLUG`: The slug of the repository.
- `REVISION`: The revision number.
- `PATH`: If path ends with `/` it will be interpreted as a directory and will return a list.

RELATED TOPICS

Using the bitbucket REST APIs

SSH Keys

The bitbucket 'SSH keys' REST endpoint provides functionality for adding, removing, and viewing SSH keys on your account.

The keys in the examples are truncated for demonstration purposes.

```
            "path": "resource.py",
```

Querying SSH Keys

```
$ curl --user mcatalbas:password https://api.bitbucket.org/1.0/ssh-keys/
[
    {
        "pk": 1,
        "key": "ssh-dss AAAAB3NzaC1kc3MA ... .atlassian.com"
    }
](
```

The command above lists all public SSH keys on your account.

Status Codes

200: Ok

On normal successful execution.

400: Bad Request

Returned if the SSH key is not valid or already registered in bitbucket.

401: Unauthorized

Returned if you have not provided your login credentials.

404: Not Found

Returned if the resource does not exist.

Adding SSH Keys

To add a new key, issue a `POST` request:

```
$ curl --request POST --user mcatalbas:password https://api.bitbucket.org/1.0/ssh-keys/ --data
"key=ssh-dss%20AAAAB3NzaC1kc3M ... .atlassian.com"
{
    "pk": 6,
    "key": "ssh-dss%20AAAAB3NzaC1kc3M ... .atlassian.com"
}
```

A public key contains characters need to be escaped before sending it as a `POST` data. So, you need to apply proper escaping (`urlencode`), if you are trying to add a key via your terminal.

Status Codes

200: Ok

On normal successful execution.

400: Bad Request

Returned if the SSH key is not valid or already registered in bitbucket.

401: Unauthorized

Returned if you have not provided your login credentials.

Deleting SSH Keys

To delete a key, issue a `DELETE` request:

```
$ curl --request DELETE --user mcatalbas:password https://api.bitbucket.org/1.0/ssh-keys/6/
```

Status Codes

204: No Content

On normal successful execution.

400: Bad Request

Returned if the SSH key is not valid or already registered in bitbucket.

401: Unauthorized

Returned if you have not provided your login credentials.

404: Not Found

Returned if the resource does not exist.

Users

You can use the bitbucket `'users'` REST resource to look up information about a user.

Overview

This is a read-only resource.

Example:

```
$ curl https://api.bitbucket.org/1.0/users/jespern/

{
    "repositories": [
        {
            "description": "",
            "followers_count": 0,
            "name": "hg-website-test",
            "resource_uri": "/1.0/repositories/jespern/hg-website-test/",
            "slug": "hg-website-test",
            "website": null
        },
    # more ...
    ],
    "user": {
        "username": "jespern",
        "first_name": "Jesper",
        "last_name": "Noehr",
        "avatar":
"https://secure.gravatar.com/avatar/b658715b9635ef057daf2a22d4a8f36e?d=identicon&s=32",
        "resource_uri": "/1.0/users/jespern"
    }
}
```

If you make an authenticated request, you will be able to see more information about the user.

```
$ curl --user username:password https://api.bitbucket.org/1.0/users/jespern/

{
    "repositories": [
        {
            "description": "",
            "followers_count": 0,
            "name": "hg-website-test",
            "resource_uri": "/1.0/repositories/jespern/hg-website-test/",
            "slug": "hg-website-test",
            "website": null
        },
    # more ...
    ],
    "user": {
        "username": "jespern",
        "first_name": "Jesper",
        "last_name": "Noehr",
        "avatar":
"https://secure.gravatar.com/avatar/b658715b9635ef057daf2a22d4a8f36e?d=identicon&s=32",
        "resource_uri": "/1.0/users/jespern/"
    }
}
```

Getting Basic User Information

Gets basic user information, including the public repositories.

```
GET /users/username/
```

Alternatively, you can use a validated email address in place of a username (e.g.
https://api.bitbucket.org/1.0/users/evzijst@atlassian.com/).

You can use the bitbucket 'newuser' REST resource to create new users.

Creating a User

This is a POST only REST resource.

Example:

```
$ curl --request POST https://api.bitbucket.org/1.0/newuser/ --data
"email=dummy@address.org&password=allyourbase&username=arebelongtous"
{
    "username": "arebelongtous",
    "email": "dummy@address.org"
}
```

Throttling

This resource is throttled to 100 calls every 30 minutes.

RELATED TOPICS

Using the bitbucket REST APIs

Getting Information about User Events

See Events.

Getting Information about Followers of a User

See Followers.

RELATED TOPICS

Bitbucket Documentation

Wiki

The bitbucket 'wiki' REST resource provides functionality for getting information from pages in the bitbucket wiki, creating new pages and updating them.

Getting the Raw Content of a Wiki Page

```
GET /repositories/USERNAME/REPO_SLUG/wiki/PAGE/
```

Parameters:

- USERNAME: The owner's username.
- REPO_SLUG: The slug of the repository.
- PAGE: The title of the page to get. Do not include the extension .wiki. If not given, the default is 'Home'.

Response:

- data: contains the contents of the wiki page
- rev: is the current revision of the wiki page, can optionally be provided when updating a pages contents

Creating a New Wiki Page

```
POST /repositories/USERNAME/REPO_SLUG/wiki/PAGE/
```

Parameters:

- USERNAME: The owner's username.
- REPO_SLUG: The slug of the repository.
- PAGE: The title of the page to get. Do not include the extension .wiki.

Use the following form parameter to specify the content of the page:

- data: The page content.

Updating a Wiki Page

```
PUT /repositories/USERNAME/REPO_SLUG/wiki/PAGE/
```

Parameters:

- USERNAME: The owner's username.
- REPO_SLUG: The slug of the repository.
- PAGE: The title of the page to get. Do not include the extension .wiki.

Use the following query string parameter to specify the content of the issue:

- data: The page content.
- rev (optional): The revision of the file before it was modified. This should be provided if want bitbucket to stop concurrent modifications to an individual wiki page. If omitted bitbucket will assume that your contents should overwrite any contents that currently exist. If you provide the rev and the wiki pages current rev does not match then your edit will fail.

RELATED TOPICS

OAuth on Bitbucket

Introduction

OAuth is a protocol through which applications can be authorized to share private data. bitbucket's OAuth support enables third-party applications to interact with repositories hosted on the site.

An IDE, for example, could sync a bitbucket repository in response to local changes, or an iPhone app could provide a custom UI for managing issues tracked on bitbucket.

Authentication

We support basic authentication and OAuth 1.0a for all authentication and authorisation purposes. See RFC 2617 & RFC 5849.

For example, the following cURL request includes basic authentication where the username and password are both 'admin':

```
curl https://bitbucket.org/api/1.0/repositories/jespern/bitbucket/issues/ --user admin:admin
```

We also support OAuth 1.0a across the API. The 3 important URLs you need to know to get started are:

- Request token: https://bitbucket.org/api/1.0/oauth/request_token/
- Authenticate: https://bitbucket.org/api/1.0/oauth/authenticate/
- Access token: https://bitbucket.org/api/1.0/oauth/access_token/

Numerous reusable libraries in many languages have been crafted for use with OAuth -- they can be found on the official oauth.net 'code' section.

Note: OAuth needs a 'consumer' defined, with both a key and a secret known to both the client and the server. If you need to register a consumer with us, please contact us on support@bitbucket.org and we'll be more than happy to accomodate your request and help you get started developing for bitbucket!

bitbucket Developer Resources

Unable to render {include} The included page could not be found.

bitbucket Developer FAQ

Here is a list of all FAQ, plus the first few lines of content for each one. Click a link to see the full text for each entry.

- What is a Slug — A repository slug is a URL-friendly version of a repository name, automatically generated by bitbucket, for use in the URL. For example, if your repository name was 'føøbar', in the URL it would become 'foobar'. Similarly, 'foo bar' would become 'foo-bar'. See the description on Wikipedia.

What is a Slug

A repository slug is a URL-friendly version of a repository name, automatically generated by bitbucket, for use in the URL. For example, if your repository name was 'føøbar', in the URL it would become 'foobar'. Similarly, 'foo bar' would become 'foo-bar'. See the description on Wikipedia.

Contributing to the bitbucket Developer Documentation

Would you like to share your bitbucket hints, tips and techniques with us and with other bitbucket users? We welcome your contributions.

On this page:

- Updating the Developer Documentation
- Adding your Tips to hg tip
- Blogging your Technical Tips and Guides – Tips of the Trade
- Hosting your Guides on bitbucket
- What about the bitbucket Product Documentation?
  - Following our Style Guide
  - How we Manage Community Updates

Updating the Developer Documentation

Have you found a mistake in the documentation, or do you have a small addition that would be so easy to add yourself rather than asking us to do it? You can update the documentation page directly. **Just sign up for a wiki username then log in and make the change**.

Adding your Tips to hg tip

Add your tips at hg tip, a repository of short, easy-to-understand tips for using Mercurial. The site is published from a repository at bitbucket. See the guide to contributing.

Blogging your Technical Tips and Guides – Tips of the Trade

Have you written a blog post describing a specific configuration of bitbucket or a neat trick that you have discovered? Let us know, and we will link to your blog from our documentation. More....

Hosting your Guides on bitbucket

You can use bitbucket as a site to publish your own documentation using a static website. A great example is Mercurial: The Definitive Guide by Bryan O'Sullivan. Another example is the TortoiseHg documentation.

See our guide to Publishing a Website on bitbucket.

What about the bitbucket Product Documentation?

Our documentation wiki contains developer-focused documentation (such as API guides, plugin and gadget development guides and guides to other frameworks) as well as product documentation (user's guides, administrator's guides and installation guides). The wiki permissions are different for each type of documentation.

- If you want to update the bitbucket API documentation, the Developer Network or other developer-focused wiki spaces, just sign up for a wiki username then log in and make the change.
- If you want to update the bitbucket product documentation, we ask you to sign the Atlassian Contributor License Agreement (ACLA) before we grant you wiki permissions to update the documentation space. Please read the ACLA to see the terms of the agreement and the documentation it covers. Then sign and submit the agreement as described on the form attached to that page.

Following our Style Guide

Please read our short guidelines for authors

How we Manage Community Updates

Here is a quick guide to how we manage community contributions to our documentation and the copyright that applies to the documentation:

- **Monitoring by technical writers.** The Atlassian technical writers monitor the updates to the documentation spaces, using RSS feeds and watching the spaces. If someone makes an update that needs some attention from us, we will make the necessary changes.
- **Wiki permissions.** We use wiki permissions to determine who can edit the documentation spaces. We ask people to sign the Atlassian Contributor License Agreement (ACLA) and submit it to us. That allows us to verify that the applicant is a real person. Then we give them permission to update the documentation.
- **Copyright.** The Atlassian documentation is published under a Creative Commons CC BY license. Specifically, we use a Creative Commons Attribution 2.5 Australia License. This means that anyone can copy, distribute and adapt our documentation provided they acknowledge the source of the documentation. The CC BY license is shown in the footer of every page, so that anyone who contributes to our documentation knows that their contribution falls under the same copyright.

RELATED TOPICS

Tips of the Trade
Author Guidelines
Atlassian Contributor License Agreement

Tips, tricks, and FAQ

Tips, tricks and answers to common questions about configuring and using bitbucket:

- How is DVCS different from other version control systems
- Displaying README Text on the Overview
- Publishing a Website on bitbucket
- Repositories in Read-Only Mode
- Why does the wrong username show in my commit messages?
- What kind of limits do you have on repository/file/upload size?
- What version of Mercurial and/or Git do you support?

Need More Help?

Try Support and other resources and the bitbucket Knowledge Base.

How is DVCS different from other version control systems

If you have been using other version control systems such as Subversion (SVN) or Perforce you should feel right at home, you will find a DVCS command set is very similar. The main difference between a central version control system and a distributed one is that the distributed system does not rely on one central server. Every person with a repository also has the full history of changes. Each repository is independent.

In Subversion, for example, each developer checks out a copy from the main server, works on changes, and commits them back to the central server. In case of conflicting changes made by other developers, the developer is notified and asked to merge the changes. In a DVCS world, things are different. Commits are local, and you can commit several dozen changes locally without ever communicating with anyone else.

Displaying README Text on the Overview

If your bitbucket repository contains a README file at the root level, bitbucket will display the contents of the README at the bottom of the repository's 'Overview' tab. bitbucket can parse and display Markdown, reStructuredText, Textile, and plain text README files.

If you have multiple README files with different file extensions, they will be displayed with the following precedence:

Filename	Format
README.rst	reStructuredText
README.md	Markdown
README.mkdn	Markdown
README.markdown	Markdown
README.text	Markdown
README.textile	Textile
README.txt	Plain text*
README	Plain text*

Note: If the file is named README.txt or README and the repository language is set to Python, it will be rendered as reStructuredText.

> ⚠ **A note about preambles**
>
> In the past, one could override the format detection by adding a special comment to the top of the file like this:
>
> ```
> -*- markdown -*-
> ```
>
> Support for preambles has been removed in favour of using the format's appropriate file extension.

Publishing a Website on bitbucket

> ℹ This is a BETA feature meaning it is not fully tested. This feature is only available with Mercurial repositories.

You can use bitbucket to host a website. For example, there is a site maintained by the bitbucket 101 tutorial that lists favorite quotes by Bitbuckians. For a more advanced example, see the TortoiseHG site.. You can only use this feature with Mercurial repositories; it is not supported with Git repositories. Additionally, this feature is based on your account's username. So, your username must be acceptable to your DNS service. Upper case characters and special characters are typically not acceptable. For example, if your account's username is happy_cat, you would need to create a new account with a username of happycat to use this feature.

To use this feature, do the following:

1. Log into your bitbucket account.
2. Create a Mercurial repo and making sure to **Name** it using the following format:
   accountname.bitbucket.org

   For example, if your username is alicejones your repo **Name** should be alicejones.bitbucket.org.

3. Clone your repo to your local system.
4. Create, add, commit, and push an index.html file to your repo source.
5. Navigate to the http://*accountname*.bitbucket.org site.
   For example, if your username is is alicejones you would navigate to http://alicejones.bitbucket.org. The system displays the HTML in the site's index.html.

Some More Technical Details

This feature turns a repo into a plain static webserver that uses the repo's root as the web root. bitbucket does not support the use of HTTPS

with this feature; you cannot log into the site. The system does not issue cookies for these domains. Server-side scripts or code are not supported.

You can add images and other media to the site. You can also include JavaScripts in your HTML pages.

Repositories in Read-Only Mode

Repositories can intermittently be in 'read-only' mode. This means that nothing can be written to the repository.

This can be for a number of reasons:

- We are upgrading our storage segments and cannot accept new data.
- We are rolling out changes that require no new writes.
- The owner of the repository needs to upgrade their account to accept a higher number of users.

Read-only mode usually only lasts for a few minutes, so if your repository has been marked read-only, try it again a little while.

Why does the wrong username show in my commit messages?

To map your username to a commit, push, and other activities, bitbucket requires that the email address you commit with matches a validated email address for your account. Both Git and Mercurial allow you to configure a global username/email and a repository specific username/email. If you do not specify a repository specific username/email values, both systems use the global default. So, for example, if your bitbucket account has a validated email address of `joe.foot@gmail.com`, you need to make sure your repository configuration is set for that username. Also, make sure you have set your global username/email address configuration to a validated email address.

If the global default is not configured or if you have not validated your email address, the committer appears as **unknown** for your bitbucket activities. Also, if you have multiple bitbucket accounts, you may mistakenly commit using configuration (global or specific) that maps to an account name you did not intend. To have the existing commits map to a different account, you can use **Admin > Username aliases** for the repository in question. Aliases are per-repository. To set up an alias for a repo, you must have **admin** rights on it.

For information about configuring your usersname or aliases, see Setting your username for bitbucket actions.

What kind of limits do you have on repository/file/upload size?

We don't place any limits on the size of your repositories, file uploads, or the number of public repositories you can have. Not on the paid plan or on the free plan. We *do* expect that you are polite and respect fair use. If you push your entire MP3 collection that is not polite or respectful to the artists.

What version of Mercurial and/or Git do you support?

You can install either or both Git and Mercurial on any of their supported operating systems. For Git, bitbucket supports 1.6.6 or later and Mercurial version 1.7 or later. If you are a developer and want to

Support and other resources

Contacting Us

- Mailing list at Google Groups: bitbucket-users
- IRC: `#bitbucket` on `irc.freenode.net`

Issue Tracking and Feature Requests

- bitbucket issue tracker

News

- bitbucket blog
- Atlassian blog

Help

- Discussions at Google Groups: bitbucket-users
- Documentation: Using your bitbucket repository, wiki and issue tracker

- Support: Atlassian Support
- Knowledge base: bitbucket Knowledge Base

API and Developer Documentation

- REST APIs, writing brokers, and OAuth

Translated Documentation

- Japanese documentation -

bitbucket Release Notes

Want to let us know about an improvement you would like to see? Please log it in our issue tracker.

Follow our blog for the latest updates. Let us whet your appetite:

Bitbucket
(Unlimited DVCS Code Hosting, Free)
Pull requests across branches
SourceTree 1.3 our FREE DVCS client now available
How Atlassian migrated from SVN to Git
Follow Up On Our Downtime Last Week
Introducing Keyboard Shortcuts
Unplanned downtime today
Mobile apps for Bitbucket
Scheduled Maintenance December 19 at 02:00 UTC
Pull requests with side-by-side diffs
Scheduled Maintenance December 5th at 0200 GMT

Full Table of Contents

This is a complete table of contents for the bitbucket documentation.

bitbucket 101

- Set up Git and Mercurial
- Create an Account and a Git Repo!
- Clone Your Git Repo and Add Source Files
- Fork a Repo, Compare Code, and Create a Pull Request
- Add Users, Set Permissions, and Review Account Plans
- Set up a Wiki and an Issue Tracker
- Set up SSH for Git
- Set up SSH for Mercurial
- Mac Users: SourceTree a Free Git and Mercurial GUI

Importing code from an existing project

Using the SSH protocol with bitbucket

- Configuring Multiple SSH Identities for GitBash, Mac OSX, & Linux
- Configuring Multiple SSH Identities for TortoiseHg
- Troubleshooting SSH Issues

Repository privacy, permissions, and more

Managing your account

- Managing Repository Users
- Managing the user groups for your Repositories
- Setting Your Email Preferences
- Deleting Your Account
- Renaming Your Account
- Using your Own bitbucket Domain Name

Administrating a repository

- Granting Groups Access to a Repository
- Granting Users Access to a Repository

Using your repository

Using your bitbucket Issue Tracker

Using a bitbucket Wiki

REST APIs, writing brokers, and OAuth

Tips, tricks, and FAQ

Support and other resources

Keyboard Shortcuts

Translated Documentation

Contributing to the bitbucket Documentation

Would you like to share your bitbucket hints, tips and techniques with us and with other bitbucket users? We welcome your contributions.

On this page:

Adding your Tips to hg tip

Add your beginner's or advanced tips at hg tip, a repository of short, easy-to-understand tips for using Mercurial. The site is published from a repository at bitbucket. See the guide to contributing.

Blogging your Technical Tips and Guides – Tips of the Trade

Have you written a blog post describing a specific configuration of bitbucket or a neat trick that you have discovered? Let us know, and we will link to your blog from our documentation. More....

Contributing Documentation in Other Languages

Have you written a guide to bitbucket in a language other than English, or translated one of our guides? Let us know, and we will link to your guide from our documentation. More....

Hosting your Guides on bitbucket

You can use bitbucket as a site to publish your own documentation as a static website. A great example is Mercurial: The Definitive Guide by Bryan O'Sullivan. Another example is the TortoiseHg documentation.

See our guide to Publishing a Website on bitbucket.Publishing Documentation on bitbucket

Updating the Documentation Itself

Have you found a mistake in the documentation, or do you have a small addition that would be so easy to add yourself rather than asking us to do it? You can update the documentation page directly.

Getting update rights on the documentation

We do ask you to sign the Atlassian Contributor License Agreement (ACLA) before we grant you wiki permissions to update the documentation space. Please read the ACLA to see the terms of the agreement and the documentation it covers. Then sign and submit the agreement as described on the form linked to that page.

Following our Style Guide

Please read our short guidelines for authors

How we Manage Community Updates

Here is a quick guide to how we manage community contributions to our documentation and the copyright that applies to the documentation:

- **Monitoring by technical writers.** The Atlassian technical writers monitor the updates to the documentation spaces, using RSS feeds and watching the spaces. If someone makes an update that needs some attention from us, we will make the necessary changes.
- **Wiki permissions.** We use wiki permissions to determine who can edit the documentation spaces. We ask people to sign the Atlassian Contributor License Agreement (ACLA) and submit it to us. That allows us to verify that the applicant is a real person. Then we give them permission to update the documentation.
- **Copyright.** The Atlassian documentation is published under a Creative Commons CC BY license. Specifically, we use a Creative Commons Attribution 2.5 Australia License. This means that anyone can copy, distribute and adapt our documentation provided they acknowledge the source of the documentation. The CC BY license is shown in the footer of every page, so that anyone who contributes to our documentation knows that their contribution falls under the same copyright.

RELATED TOPICS

Tips of the Trade
Author Guidelines
Atlassian Contributor License Agreement

Tips of the Trade

A number of enthusiastic and generous bloggers and authors have written some excellent guides to bitbucket. Below are links to some of them.

> ⚠️ **Please be aware that these are external blogs and articles.**
>
> Most of the links point to external sites, and some of the information is relevant to a specific release of bitbucket. Atlassian provides these links because the information is useful and relevant at the time it was written. Please check carefully whether the information is still relevant when you read it.

On this page:

- Learn Mercurial one bite-sized tip at a time
- Various posts about Mercurial
- A Guide to Branching in Mercurial
- A Git User's Guide to Mercurial Queues
- Setting up SSH access to bitbucket on Windows with PuttyGen

Mercurial Hints and Tips
Learn Mercurial one bite-sized tip at a time • By: Everyone, on the 'hg tip' website. • About: Hints and tips about using Mercurial • Date: All the time • Related documentation: bitbucket 101 Various posts about Mercurial • By: Bryan O'Sullivan, on his blog 'teideal glic deisbhéalach' • About: Hints and tips about using Mercurial • Date: Frequently • Related documentation: Creating a Repositorybitbucket 101

Branching and Forking

A Guide to Branching in Mercurial

- By: steve losh, on his blog
- About: A guide to the concepts behind the branching models in Mercurial
- Date: 30 August 2009
- Related documentation: Forking a bitbucket Repository

Explaining Mercurial to Git Users

A Git User's Guide to Mercurial Queues

- By: steve losh, on his blog
- About: Trying to explain MQ in a way that git users might find more understandable
- Date: 10 August 2010

Getting Technical

Setting up SSH access to bitbucket on Windows with PuttyGen

- By: alinium, on posterous.
- About: Setting up SSH access to a bitbucket Mercurial repository on Windows
- Date: 30 November 2010
- Related documentation: Using the SSH protocol with bitbucket

✅ **Have you written a technical tip for Bitbucket?**

Add a comment to this page, linking to your blog post or article. We will include it if the content fits the requirements of this page. Or you can sign up to edit the bitbucket documentation yourself, including this page.

✅ **Feedback?**

Your first port of call should be the author of the linked blog post. If you want to let us know how useful (or otherwise) a linked post is, please add a comment to this page.

bitbucket Documentation in Other Languages

Below are some links to bitbucket documentation written in other languages. In some cases, the documentation may be a translation of the English documentation. In other cases, the documentation is an alternative guide written from scratch in another language. This page presents an opportunity for customers and community authors to share documentation that they have written in other languages.

⚠️ **Please be aware that these are external guides.**

Most of the links point to external sites, and some of the information is relevant to a specific release of bitbucket. Atlassian provides these links because the information is useful and relevant at the time it was written. Please check carefully whether the information is still relevant when you read it. The information in the linked guides has not been tested or reviewed by Atlassian.

On this page:

- No guides yet

None
No guides yet
We do not yet have any guides to link here. Be the first to suggest one!

Adding Your Own Guide to this Page

Have you written a guide for bitbucket in another language? Add a comment to this page, linking to your guide. We will include it if the content fits the requirements of this page.

Giving Feedback about One of the Guides

If you have feedback on one of the guides listed above, please give the feedback to the author of the linked guide.

If you want to let us know how useful (or otherwise) one of these guides is, please add a comment to this page.

Other Sources of Information

bitbucket documentation
Atlassian website
Atlassian blog

Keyboard Shortcuts

bitbucket comes with keyboard shortcuts you can use to quickly access pages and features. To use these shortcuts, the bitbucket page must be active in your browser (Firefox, Internet Explorer, Safari, etc.). So, for example, in Firefox to make a page active you click its tab. Once you have an active window you are free to use any of the shortcuts. Most shortcuts are a combination of keys where you press one key and then the next in sequence, such as **g** then **i** to go to your inbox. Some shortcuts are a single key press, such as **/** (forward slash) to put your cursor in the search field.

Don't like keyboard shortcuts? You can disable them on your **Account settings** page. Just uncheck the box marked **Keyboard shortcuts**.

Global Navigation

These keys give you access to the global links in the bitbucket header as well as the "important" actions like creating a repository or searching.

Key or Combination	Action
c then **r**	Create repository.
i then **r**	Import repository.
g then **d**	Go to dashboard.
g then **a**	Go to account settings.
g then **i**	Go to inbox.
g then **e**	Explore bitbucket.
g then **r**	Go to **My repositories**.
/	Focus the site search. Puts your cursor in the search field.
esc	Dismisses the help dialog or removes the focus from a form field.
u	Goes back up the stack you just went down with the shortcuts. Like the back button in a browser, this takes you back through the bitbucket pages you just paged through.

Repository Navigation

Use these to navigate the listings in the repository view.

Key or Combination	Action
r then **o**	Repository overview.
r then **d**	Repository downloads.
r then **q**	Repository patch series.
r then **p**	Repository pull requests.
r then **s**	Repository source.
r then **c**	Repository commits.
r then **w**	Repository wiki.
r then **i**	Repository issues.
r then **a**	Repository administration.

Repository List Navigation

Use these to navigate the listings in the repository view.

Key or Combination	Action
j	Select next item.
k	Select previous item.
o or **return**	View selected item.

Key or Combination	Action
j	Select next item.
k	Select previous item.
o or **return**	View selected item.