AccuRev

GitCentric™

User's Help

Version 2015.1

Revised 2/17/15

Table of Contents

Preface vii
- Audience vii
- Using This Book vii
- Typographical Conventions viii
- Contacting Technical Support viii

1. Concepts and Overview 1
- What is AccuRev GitCentric? 1
- Where to go for More Information 1
  - A Note About Terminology 2
- Basic Architecture 3
- GitCentric Users 4
- GitCentric Administrators and Mapping Branches to Streams 4
  - Basic Rules 4
  - Mapped Behavior 5
  - Best Practices When Planning Your Installation 5
  - Keeping Git Merges and AccuRev Promotes in Sync 5
  - Best Practice for Keeping Merges and Promotes in Sync 7
  - Scenarios for Mapping Multiple Repositories to a Single Stream 7
  - Multiple Repos and Multiple Servers 8
  - AccuRev Replica Server Restriction 9
- Configuring for Security 10
  - Configuring Multiple Git Repos with AccuRev and EACLs 11
- Getting Started 11

2. How to 13
- Procedures for All Users 13
- Get Started 13
  - Define and Display a Site Header or Footer for Gerrit Code Review 18
  - Configure the Clone for Code Review (Optional) 20
  - Configure the Clone for Direct Push 20
  - Troubleshoot Git Clone Issues 20
  - Information Displayed on the Commits Page 22
  - Starting from the Commits Page 23
  - Starting from the Source Tree Page 24
  - Information Displayed on the Source Tree Page 24
  - Configure GitCentric 29
  - Remove a Branch 31
  - Remove a Repository 31
  - A Note About AccuRev Depots 31
  - Import a Snapshot of the Latest Heads into AccuRev 32

General Procedure for Setting ACLs 34
Configuring ACLs for Code Review 36
Configuring GitCentric ACLs for Direct Push 38
Avoid Git Reserved Name for AccuRev Elements 39
Specifying the AccuRev Server Connection 39
Mapping the Branch to the Stream 40
Specifying the Commit Message Format 43
Troubleshooting Change Package Errors 45
Registering an AccuRev Server 46
Configuring the AccuRev Server 47
Configure Multiple AccuRev Servers 48
To View and Access Groups 49
Add a Group 49
Add a Member to a Group 50
AccuRev Groups 51
Allowing Self-Reviews 52
Disabling Code Review 53
Overview 53
Registering the GitCentric Bridge SSH Key with Gerrit 53
Modifying the replication.config File 54
AccuRev to Git 54
Git to AccuRev 55
Gerrit gc Syntax 55
cron job Examples 56

3. My Account 57
Opening the My Account Page 57
Menu Options 57
Profile 57
Watched Repositories (Projects) 58
Contact Information 59
Public Keys 59
AccuRev Servers 59
HTTP Password 60
People 60

4. Code Review 61
Opening the Code Review Page 61
Overview of Gerrit Code Review 61
Differences from Standalone Gerrit Code Review 61
Code Review for Users of Differing Backgrounds 62
Gerrit Code Review and AccuRev Mappings 63
Troubleshooting 63

5. Administration 65
Opening the Administration Page 65

Repositories 66
To Create a New Repository 66
To Configure an Existing Repository 66
Support for Hooks 73
People 73
AccuRev Servers 74
A Note about the CLI Path Setting 74

A. The kandoMaintain Utility 75
Using kandoMaintain 75
Backup and Restore 75
kandoMaintain Command Reference 75
Commands 75
Options 78
GitCentric Bridge Configuration Settings 80
Examples 80

B. Backup and Restore 83
Commands for Backup and Restore 83
Best Practices 83
Backing Up GitCentric 84
What Gets Backed Up? 84
How to Back Up GitCentric 84
Restoring GitCentric 85
Caution: Restore Overwrites Existing GitCentric Installations 85
Prerequisites 85
What Gets Restored? 85
How to Restore GitCentric 85
Next Steps 86
Restore Scenarios 87

C. Command-Line Reference 89
Basic Syntax 89
Spaces and Quoting 89
CLI Example 97

D. GitCentric Glossary 99

Preface

This document serves as both the on-line help and the User's Guide for AccuRev GitCentric. This documentation covers both GitCentric end user and administrator audiences. GitCentric provides most functionality through a Web UI, but also provides three administrative CLI commands.

Audience

This document is intended for GitCentric end users and administrators. End users are assumed to be familiar with Git source control, and possibly Gerrit Code Review. Administrators are assumed to be familiar with these topics and also with Linux operating systems, as well as AccuRev and AccuWork concepts.

Using This Book

This book assumes you are familiar with your operating system(s) and their commands, as well as with AccuRev, AccuWork, and Git.

The following table summarizes the chapters and appendixes in this book.

Chapter	Description	Audience
Chapter 1 Concepts and Overview	Introduces basic GitCentric concepts and architecture.	End users and administrators
Chapter 2 How to...	Provides a series of common tasks that you perform to configure and use GitCentric.	End users and administrators
Chapter 3 My Account	Provides the ability to register yourself with GitCentric, and to set your preferences and contact information.	End users and administrators
Chapter 4 Code Review	Gives you access to optional third-party Gerrit Code Review functionality.	End users and administrators
Chapter 5 Administration	Summarizes the features for configuring and maintaining repositories, groups, and AccuRev servers, including security configuration. Provides links to conceptual and task-based sections of this document.	Administrators
Appendix A The kandoMaintain Utility	Describes the command line **kandoMaintain** utility for upgrading and administering the GitCentric database.	Administrators
Appendix B Backup and Restore	Summarizes procedures and best practices for backing up your GitCentric-related repositories and database files.	Administrators

Chapter	Description	Audience
Appendix C Command-Line Reference	Summarizes the syntax and usage of the GitCentric administration commands, typically used for scripting administrative functions. **Note:** Although these commands are generally considered administrator commands, any registered user may use the `ls-repo` command to view the repos to which they have access.	Administrators
Appendix D GitCentric Glossary	A list of GitCentric-related terms and their definitions.	End users and administrators

Typographical Conventions

This book uses the following typographical conventions:

Convention	Description
`blue sans-serif`	Used for sample code or output.
bold	Used for command names, and button names in the GitCentric user interface
light italic	Used for emphasis, book titles, and for first use of important terms
blue italic	Identifies a hyperlink (to a page or Web URL, for example)

Contacting Technical Support

Micro Focus offers a variety of options to meet your technical support needs as summarized in the following table.

For	Visit
Information about technical support services	*http://supportline.microfocus.com/*
Information about platforms support	*http://supportline.microfocus.com/prodavail.aspx*
Product downloads and installations	*http://supportline.microfocus.com/websync/productupdatessearch.aspx*
Product documentation	*http://supportline.microfocus.com/productdoc.aspx*
SupportLine phone numbers, listed by country	*http://www.microfocus.com/about/contact/support/assistance.aspx*

When you contact Micro Focus technical support, please include the following information:

- The version of AccuRev and any other AccuRev products you are using (AccuSync or GitCentric, for example).
- Your operating system.
- The version of relevant third-party software (if you are using AccuSync, for example the version of your ITS).

- A brief description of the problem you are experiencing. Be sure to include which AccuRev interface you were using (Web user interface, Java GUI, or CLI), any error messages you received, what you were doing when the error occurred, whether the problem is reproducible, and so on.
- A description of any attempts you have made to resolve the issue.
- A simple assessment of how the issue affects your organization.

1. Concepts and Overview

This chapter provides an introduction to the concepts behind AccuRev GitCentric.

What is AccuRev GitCentric?

AccuRev GitCentric is a bridge between two worlds:

- the open source Git revision control system
- the Enterprise-capable AccuRev source control management (SCM) system

Git is popular with many developers for its simplicity and speed. However, professional enterprises need more control and scalability for their large investment in intellectual property. Many developers prefer AccuRev's interface; release engineers require AccuRev's stream architecture for capturing the exact revisions needed for a build environment, and managers require AccuRev's TimeSafe auditablility and its ACL-based security.

How does an organization meet these differing requirements and preferences?

AccuRev GitCentric provides enterprises with the solution: developers who embrace Git can continue using Git the same way they do today, while other development groups, release and test engineers, and managers who require AccuRev's power, functionality, and security can work together via the AccuRev GitCentric interface. GitCentric also incorporates the open source Gerrit Code Review package for optional code review functionality.

In the most basic terms, Git repository branches are mapped to AccuRev streams, so that the two stay in sync. Changes to the Git repository get transmitted to the AccuRev depot, and changes in the AccuRev depot get transmitted to the Git repository. Updates happen automatically. Git users do not need change their work practices for the AccuRev environment, since GitCentric is transparent to them.

GitCentric functionality applies to two audiences:

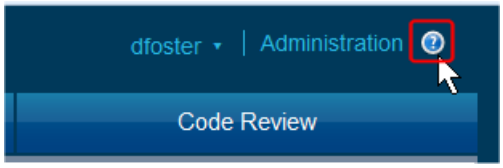
- Regular users can register themselves, maintain their preferences, and (if implemented at their site) access the optional code review functionality.
- Administrators can configure and maintain repositories, their security, and their branch mappings to AccuRev streams.

Where to go for More Information

When using GitCentric, the documentation you need depends on what component you are using.

- For GitCentric installation, see the AccuRev *GitCentric Installation and Release Notes*.

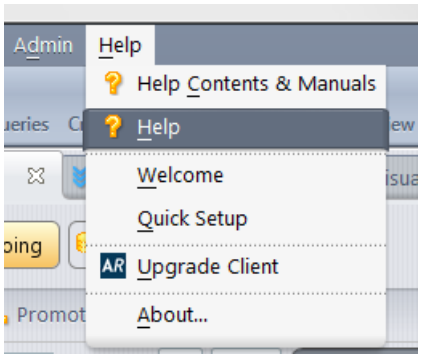
- For GitCentric administration and use, you're already here: the AccuRev *GitCentric User's Help* (available in both PDF and HTML) from the GitCentric web interface:



- For help with Gerrit Code Review, see the Gerrit Code Review documentation here:

  http://gerrit-documentation.googlecode.com/svn/Documentation/2.7/index.html

- For Git documentation, you can use `git help` from the command line. You can also use a search engine to locate several good tutorials and discussions available on the web.
- For AccuRev documentation, access the complete documentation set in HTML and PDF from the `Help Contents & Manuals` menu in the AccuRev GUI:



The *AccuRev Installation and Release Notes* are available from the Micro Focus Product Documentation page here: *http://supportline.microfocus.com/productdoc.aspx*

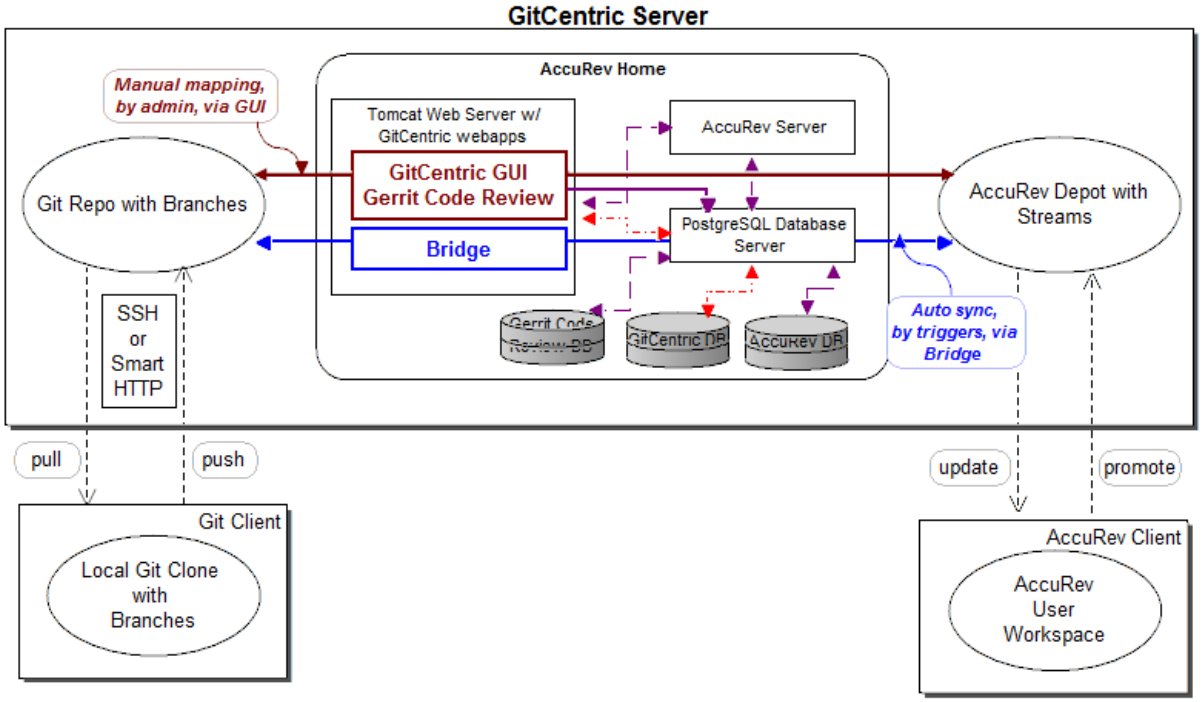
A Note About Terminology

Because GitCentric spans the environments of AccuRev, Git, and Gerrit Code Review, you may encounter some conflicts or overlaps in terminology. For example, Gerrit Code Review often uses the term "project" to refer to a repository. AccuRev uses the terms "repository" or "repo" when referring to a repository, and reserves the term "project" for referring to specific Gerrit functionality, or when referring to some kind of planned or defined undertaking. See *Appendix D GitCentric Glossary* for definitions of terms used in GitCentric.

Basic Architecture

The following diagram shows the basic configuration of a very simple GitCentric environment where the Git repository, Tomcat web server, and AccuRev server with databases for both GitCentric and AccuRev reside on the same server:

Figure 1. Basic Architecture



In this simple configuration, the GitCentric server hosts:

- a Tomcat web server configured with two GitCentric apps (a bridge and an administrative GUI)
- an AccuRev server which includes a PostgreSQL database, and which functions as both the GitCentric AccuRev server and a production SCM AccuRev server with a trigger (server_master_trigger) to notify GitCentric of changes in AccuRev
- a GitCentric installation directory, which generally (but not necessarily) contains a storage area for one more Git repositories which include triggers (pre-receive and update) and configuration files. (The triggers are Git hooks that get copied to your GitCentric repos and which help keep AccuRev synchronized with your repo. Search the web for more information about Git hooks.)

In this simple environment, the AccuRev server hosts both the AccuRev depot containing the streams that are mapped to branches in the Git repo, along with the database server for both GitCentric and AccuRev, and another trigger that keeps your repo synchronized with AccuRev. In fact the Tomcat server is also a part of the AccuRev installation, under the `<ac_home>/WebUI` directory.

For the sake of this discussion, we will assume the simple configuration in which the Git repo and the AccuRev server (which includes the Tomcat server, the GitCentric database, and the production SCM database) all reside on the same machine.

Regardless of the complexity of the installation, the basic GitCentric process remains the same: Git users push and pull files between their local Git repos and the GitCentric Git repo. AccuRev users promote and

update files between their workspaces and the AccuRev Server. And GitCentric keeps the branches in the Git repos synchronized with streams in the AccuRev servers.

GitCentric Users

All GitCentric users -- whether they are Git developers or GitCentric administrators -- use GitCentric to self-register their accounts and their SSH public keys, and to maintain their account preferences. If code review is configured, then Git developers will also access Gerrit Code Review through GitCentric.

However, many GitCentric features are geared toward administrators so that they configure AccuRev servers and their mappings to Git repository branches.

GitCentric Administrators and Mapping Branches to Streams

A GitCentric administrator uses the GitCentric GUI (and optionally the GitCentric CLI commands) to configure the mapping between Git branches and AccuRev streams. A Git repository ("repo") is associated with a single AccuRev server, and with an AccuRev username (the "service account") for performing GitCentric administrative tasks. Once this association is defined, a GitCentric administrator can proceed to map Git repo branches to AccuRev streams on that server. The directory within a stream that is mapped to a Git branch is called the "mount point".

After the mapping has been done, the GitCentric bridge webapp uses triggers to automatically keep the Git repo and the AccuRev depot in sync, while the GitCentric GUI gives you the ability to create repos, map branches to streams, set ACLs, etc.

Note that you can have multiple GitCentric AccuRev user accounts:

- administrative users who interactively log into the GitCentric GUI (and any related AccuRev servers), for manual administrative tasks
- "service account" users, accessed internally by the GitCentric bridge, for automatically syncing the Git repos and their mapped AccuRev streams

Service account users must be defined (either as an individual user, or as a group) in the AccuRev server acserver.cnf file(s), using the ASSIGN_USER_PRIVILEGE setting in AccuRev 5.4 or later. **Note:** Best practice is to assign this to a group, so different members of the group can have different access rights. For example, one member of the group might be from an off-shore organization, with different access rights than another member of the group.

- The syntax for the entry in acserver.cnf is
  `ASSIGN_USER_PRIVILEGE = <user_or_group_name>`
- If you specify multiple ASSIGN_USER_PRIVILEGE settings, only the first one is honored.

Basic Rules

Regardless of how many Git repositories or AccuRev servers you configure, when you associate a repo with a server, you map them at the branch and stream level: a Git branch is mapped to an AccuRev stream.

- You can map multiple Git branches (and multiple Git repositories) to a single AccuRev stream.
- You can map a Git branch to any directory within an AccuRev stream

- You cannot map a Git branch to multiple AccuRev streams.
- You cannot map a Git branch to the root stream in an AccuRev depot. (If you need to do this, simply create a new stream off the root, and then map to that.)

Mapped Behavior

If you push a file to the repository and it would require merging with the version of the file on the mapped AccuRev stream, GitCentric cancels the push and informs the user that he or she must pull the latest changes, merge and retry the push, the same way Git always handles these situations.

When you first map a Git branch to an AccuRev stream, you can specify whether the files in Git or on AccuRev take precedence.

Since AccuRev streams inherit versions from their backing streams, it is important to remember that if any element is promoted to a backing stream above a stream that is mapped to a repo, that element will automatically be pushed to the repo from AccuRev.

Best Practices When Planning Your Installation

GitCentric reflects the flexibility of both the Git and AccuRev environments, and it is possible to configure your repositories and streams in countless ways. However, Borland recommends keeping the following thoughts in mind:

- Git repositories are smaller, coherent groups of functionality. You do not merge files, you merge the whole repository.
- AccuRev depots tend to be large sets of files related to entire products, or multiple products.

Therefore, you probably do not want to create a repository that maps to an entire depot. You want your Git branches to map to directories within AccuRev streams that contain smaller, independent sections of functionality.

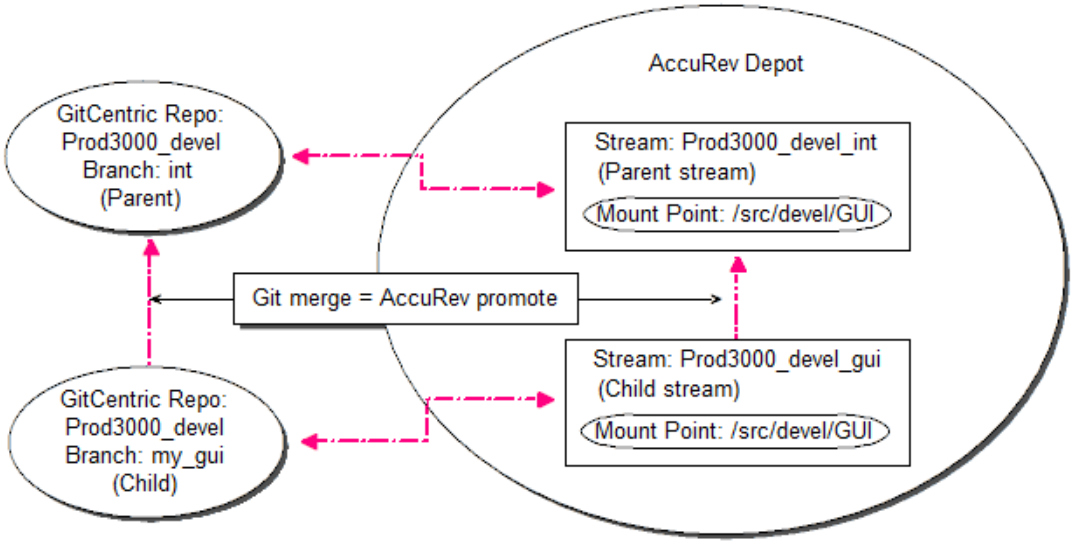
As with any new tool, it is important that you first understand what your current process is. If you are an existing Git shop, make sure that you have a clear picture of how your repositories and branches are configured, and what your workflow is. In evaluating this, you may find that you need to better define your current environment. If you have never gone through this exercise, consider searching for the following terms on the web: “git”, “workflow”, and “model”. This will point you to some good discussions about successful Git implementations. Once this is done, then you will be in a better position to decide how to map your branches to AccuRev streams.

Also, remember that both Git and AccuRev bring different strengths to your organization. Git provides a convenient, distributed version control system to your development end users. AccuRev provides powerful, centralized version control for users such as build administrators, release engineers, product managers, etc. Git branches can be somewhat transient -- if you start to develop something on a branch and then change your mind, you can delete the branch. AccuRev is TimeSafe™ -- all versions and transactions are captured permanently in the AccuRev database. When carefully planned, mapping between these environments means that GitCentric provides the best of both worlds to your organization.

Keeping Git Merges and AccuRev Promotes in Sync

When you configure a Git repository to work with AccuRev through GitCentric, you map a Git branch to an AccuRev stream. If you wish to take advantage of AccuRev **promote** operations, you map the parent AccuRev Stream to one branch, and the child AccuRev stream to another branch.

Figure 2. Git Merges and AccuRev Promotes



If you configure things correctly and follow a few basic rules, then:

- Merges between the mapped branches in Git will result in an AccuRev "promote" between the corresponding streams.
- Promotes from the child stream to the parent stream in AccuRev will result in a merge between the corresponding branches in the Git repository if the content under the mount point is empty in the child stream. (That is, the merge will happen if the child stream is completely inherited so that it is the same as the parent at the mount point).

Note that the desired condition when these operations are completed is for the "default group" in AccuRev stream "C" to be empty. (The "default group" is the set of elements or files in a stream that are under current development. A file is removed from a stream's default group when that file is either **promote**d or "**revert to back**"ed (or **purge**d). See the AccuRev documentation for more details.)

To ensure that this happens reliably and predictably, configure your system so that:

1. One stream is the parent of another stream, and each of these streams are mapped to branches in the Git repository. In this example, the parent stream is called "P" and the child stream is called "C"
2. There is no timestamp on AccuRev stream "C". (A timestamp will prevent a stream default group from being cleared.)
3. The mount point must be the same in each AccuRev stream for the branches that are mapped to them. (If one branch is mapped to one mount point in one stream, and the other branch is mapped to a different mount point in the other stream, the files will never line up.)
4. Any AccuRev rules that are applied to the C stream are also applied to the P stream. (For example, avoid include/exclude rules one stream that cause that cause its contents to differ relative to the other stream. And do not use cross-links on writable elements.)

From a Git viewpoint, this ensures that the branches point to the same commit graphs when their contents are identical. From an AccuRev viewpoint, this ensures that the default groups in the streams get cleared out when appropriate, and do not keep growing to include (potentially) all files in the system.

Best Practice for Keeping Merges and Promotes in Sync

Borland recommends that you merge into the child branch first, and then fast forward merge the parent branch. If you merge into the child branch first, AccuRev will reflect this by performing a promote from the child stream to the parent stream. However, if you merge into the parent branch first, AccuRev will achieve the desired "zero default group" state by performing an AccuRev **purge** on the child stream.

The Git commands for merging into the child branch are:

1. `git checkout C`
2. `git merge P`
3. `git checkout P`
4. `git merge C` (This will be a fast-forward merge. Existing merge commit will be reused.)
5. `git push`

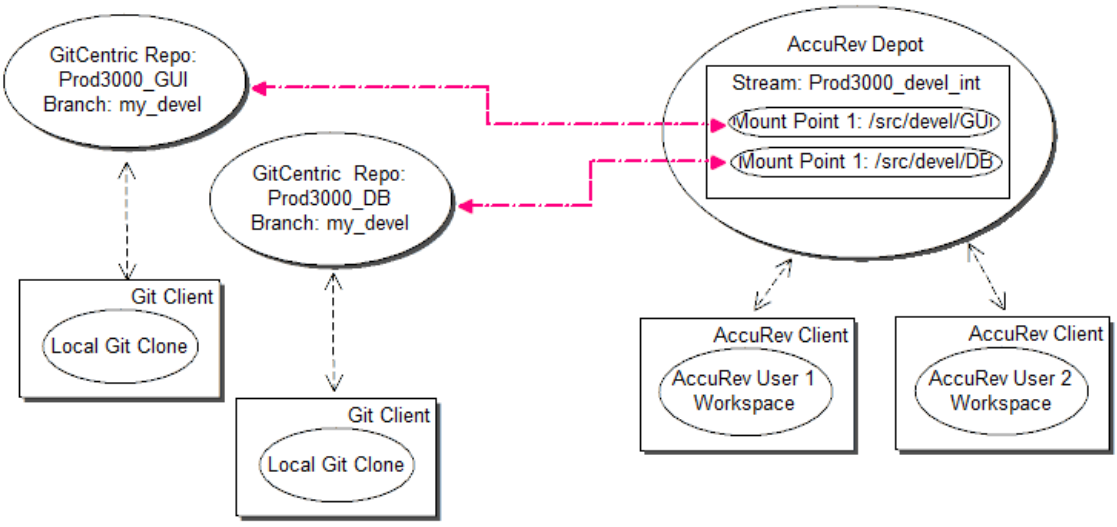
Scenarios for Mapping Multiple Repositories to a Single Stream

There are two common reasons for mapping two different Git repositories to the same AccuRev stream:

- Project-based
- Security-based

Project-based

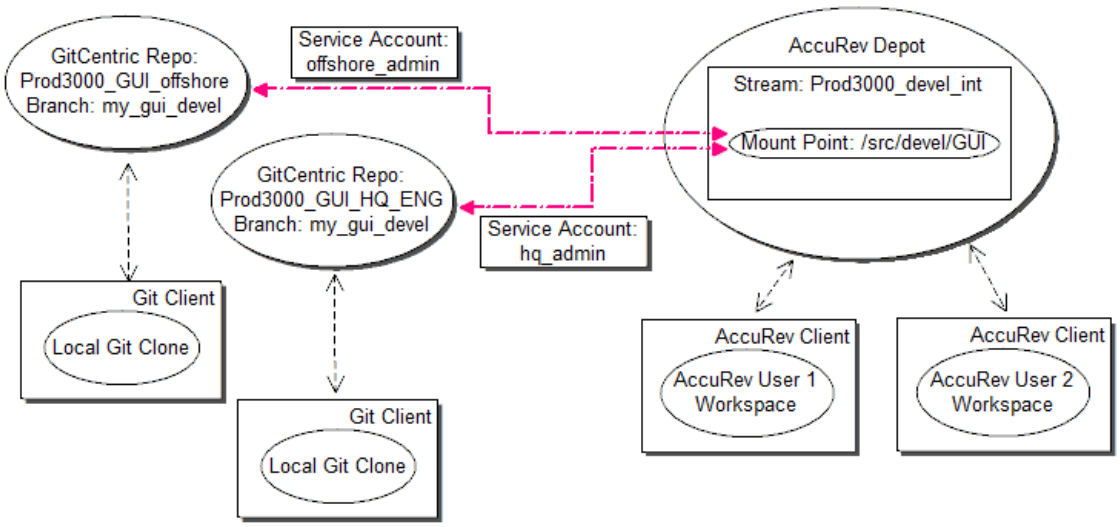
Figure 3. Project-based (same stream, different mount points)



In this case, you might have two different parts of a product in two different directory structures in the same stream, such as a GUI development tree and a database development tree. In the Git environment, you could have the GUI work being done in one repository and the database work being done in another. By mapping the branches in these repositories to the correct mount points in the same stream, you can keep the work separate. (Keeping the branch names consistent across repos will be helpful if you need to make branch-mapping changes en masse using the **children-of** option in the SSH **config-branch** CLI command.)

Security-based

Figure 4. Security-based (same stream, same mount point, different service account)



In this case, you could have two different sets of users with different access privileges accessing the same files. Privileged Git developers in corporate headquarters could have one repo mapped to the mount point with one service account having "lenient" AccuRev ACLs (see below). Less privileged off-shore contract developers could have a different repo mapped to the same mount point with a different service account having much more restrictive AccuRev ACLs. See *Configuring for Security* on page 10 for more information about ACLs and security.

Also, by mapping branches from different repos to a single AccuRev stream, you can automate the process of updating repos with changes: when a change gets pushed to a repo that is mapped to an AccuRev stream, that change gets propagated to all other branches that are mapped to that stream (assuming that the AccuRev ACLs allow a repo to "see" the changed file).

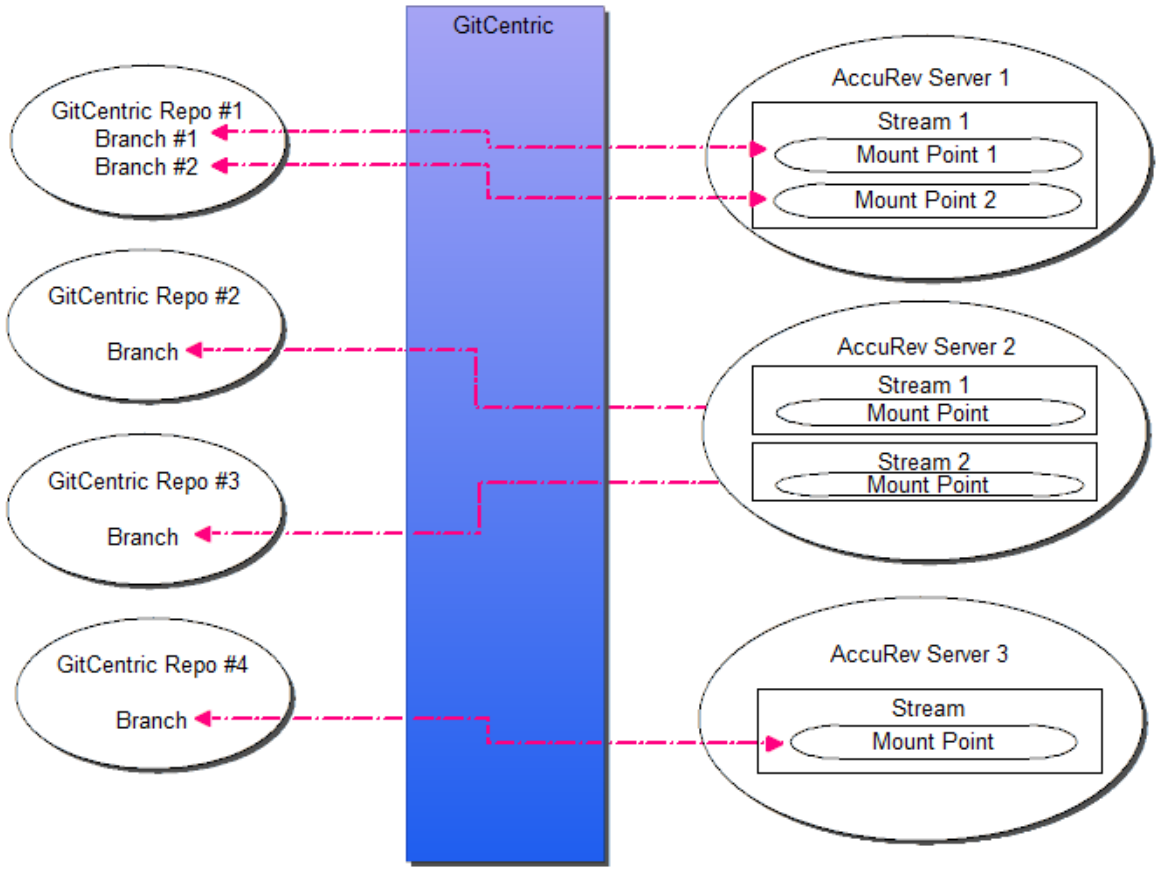
Multiple Repos and Multiple Servers

Figure 1 above illustrates a simple case of a single Git repository being associated with a single AccuRev server through GitCentric. *Figure 3* and *Figure 4* above illustrate multiple repositories being mapped to a single AccuRev server.

However, GitCentric can also configure multiple Git repositories with multiple AccuRev servers. For example, you could have one repo associated with one AccuRev server, and two other repos associated with a different AccuRev server. (However, a single repo cannot be associated with multiple AccuRev servers. And having multiple GitCentric servers configured with the same AccuRev server is not supported.)

If you choose to configure multiple AccuRev servers, AccuRev strongly recommends that you use the same GitCentric administrator user and password for all AccuRev servers to avoid the need to constantly log in and out as you move between servers.

Figure 5. Multiple AccuRev Servers



Note that if you choose, you can administer your Git repos with GitCentric and not have their branches mapped to any AccuRev streams at all. You do need to associate a Git repository with an AccuRev server for security purposes, but you do not need to have its branches mapped to use GitCentric.

AccuRev Replica Server Restriction

If your AccuRev installation has one or more replica servers, you should always try to associate GitCentric with the AccuRev master (if possible), and NOT a replica. Even if your GitCentric installation is physically located in close proximity to a replica server, you should try to configure GitCentric to work with the remote master AccuRev server, not the geographically closer replica server. (This may not be possible if your installation has implemented firewalls and VPN or some other kind of security that prevents you from communicating directly with a remote master server.)

Configuring for Security

As touched upon in *Figure 4* on page 8, one of the advantages of using AccuRev GitCentric in a Git environment is that you can use both GitCentric and AccuRev security features to control access to files, a feature not natively provided by Git.

GitCentric makes use of two kinds of Access Control Lists ("ACLs"):

- GitCentric "group-based" ACLs, which define access to Git repositories.
- AccuRev Element ACLs, which define permissions on the AccuRev server down to the individual element level.

Using these two types of ACLs, you can approach GitCentric security in two ways:

- Using GitCentric group-based ACLs to specify allow and deny privileges on a repository ("project") basis.
- Mapping multiple repos as different "views" on AccuRev-controlled elements. The AccuRev-controlled elements may optionally be secured with AccuRev Element ACLs (EACLs).

GitCentric Group-Based ACLs

On the Git side, you define group-based ACLs and apply them to repositories, to control what kind of access group members have to GitCentric-controlled Git repos and branches. (GitCentric group-based ACLs are different than -- and should not be confused with -- AccuRev Element ACLs or "EACLs", which control access to files on the AccuRev server.)

GitCentric provides five system or pre-defined internal or groups:

- Administrators (internal)
- Anonymous Users (system)
- Non-interactive Users (internal)
- Repository ("Project") Owners (system)
- Registered users (system)

You can define more groups as necessary.

GitCentric is installed with a basic set of ACLs on a special, system-defined project named "All-Projects", from which all repos inherit their base set of ACLs. These basic ACLs are set to be highly secure, so you will need to customize them for your site before your users can use GitCentric.

The general topic of group-based ACLs is beyond the scope of this document, so you will need to learn about them from the Gerrit Code Review documentation referenced below, but at a very high level:

Every user account is a member of one or more groups, and access and privileges are granted to those groups. You cannot grant access rights to individual users.

Access rights are then assigned to these groups per repo (or "project"). Access rights granted to parent repos are inherited by child repos. Access rights defined for the All-Projects project are inherited by all other projects.

For information about creating and configuring GitCentric group ACLs from the GitCentric UI, see *Configure Access Rights (ACLs) for a Repo* on page 34.

Because GitCentric group ACLs are derived from Gerrit Code Review, they are documented in detail in the Gerrit Code Review documentation. See the *Access Controls topic* in the Gerrit Code Review documentation for more information.

Configuring Multiple Git Repos with AccuRev and EACLs

In AccuRev, it is a fairly common practice to configure depots and files with ACLs so that only certain users can access them. For example, assume that you hire an off-shore contract company to develop code for an optional feature to your main product line. You might want to give staff in corporate headquarters access to all files and directories, while restricting access of the off-shore team to just those files and directories that they need to get the job done.

By setting up ACLs in the AccuRev environment, and then mapping Git repositories and branches to these AccuRev depots and streams, you can give the off-shore team access to just the repo containing their files, while giving your domestic teams access to the repo that contains all your files (see *Figure 4* on page 8).

For information about setting up ACLs in the AccuRev environment, see the following AccuRev documentation:

- *On-Line Help Guide*: Chapter 8, “Security”
- *Administrator’s Guide*: Chapter 9, “AccuRev Security Overview”
- *CLI User’s Guide*: **eacl**, **setacl**, **lsacl**, and **mkuser** command descriptions in Chapter 2, “AccuRev Command Line Reference”

Getting Started

Once you have completed this chapter, and after you have installed GitCentric as described in the AccuRev *GitCentric Installation and Release Notes*, you should proceed to *Get Started* on page 13 in *Chapter 2 How to...* to learn how to configure your GitCentric environment and use it for the first time.

2. How to...

This chapter describes how to perform basic tasks with the AccuRev GitCentric GUI. (For an example of performing some of these tasks with CLI commands, see *CLI Example* on page 97.)

For procedures that apply to all users, proceed to the next section.

For procedures that apply only to Administrator users, see *Procedures for Administrators Only* on page 28.

Procedures for All Users

The procedures within this section apply to both regular users and administrators.

Table 1. Summary of Common GitCentric Procedures for ALL Users

To:	Go to page:
Get Started	*13*
Create an SSH key	*14*
Log In to GitCentric	*14*
Register with GitCentric	*16*
Set Preferences	*18*
Generate an HTTP Password	*19*
Create a Clone From a GitCentric Repository	*19*
View Commit History	*21*
Review a Commit's Files	*23*
Compare Branches	*25*
Switch Between Gerrit Code Review and GitCentric	*26*

Get Started

This section assumes that you have completed the basic installation steps described in the AccuRev *GitCentric Installation and Release Notes*.

Now you will perform some configuration steps and actually use the GitCentric GUI for the first time.

> **Note:** Administrators -- Before you begin to associate Git repositories and branches with AccuRev depots and streams, you must have a solid understanding of your current Git and AccuRev processes. Please be sure to read through *Chapter 1 Concepts and Overview*, particularly *GitCentric Administrators and Mapping Branches to Streams* on page 4, *Mapped Behavior* on page 5, and *Best Practices When Planning Your Installation* on page 5.

Create an SSH key

This section applies to all GitCentric users, whether you are an administrator or an end user.

> **Note:** General SSH documentation is beyond the scope of this document. We recommend that you use your favorite search engine to find information about the topic. However, GitCentric administrators should know that the GitCentric installer now includes an SSHD daemon, and the product now includes a self-registration feature. This means that you no longer need to manually install and configure an SSH server, and you no longer need to create and register SSH keys for your users.

1. If you know what an SSH public key is, and you know that you have one, and you know where your public key file is, you are all set! You can skip ahead to the next section.
2. If you need to generate a key, use the following syntax:

   `ssh-keygen -t rsa -C <your_email>@<your_domain>`

   Note: If you are an end-user on a Windows machine, you should install a Git Windows client such as msysGit, which includes `ssh-keygen`. See `http://msysgit.github.com` for more information.

Make note of where your public key is stored. Typically it is in `~/.ssh/id_rsa.pub` (or `C:\users\<youraccount>\.ssh\id_rsa.pub` on Windows). You will need it when you first log in to GitCentric.

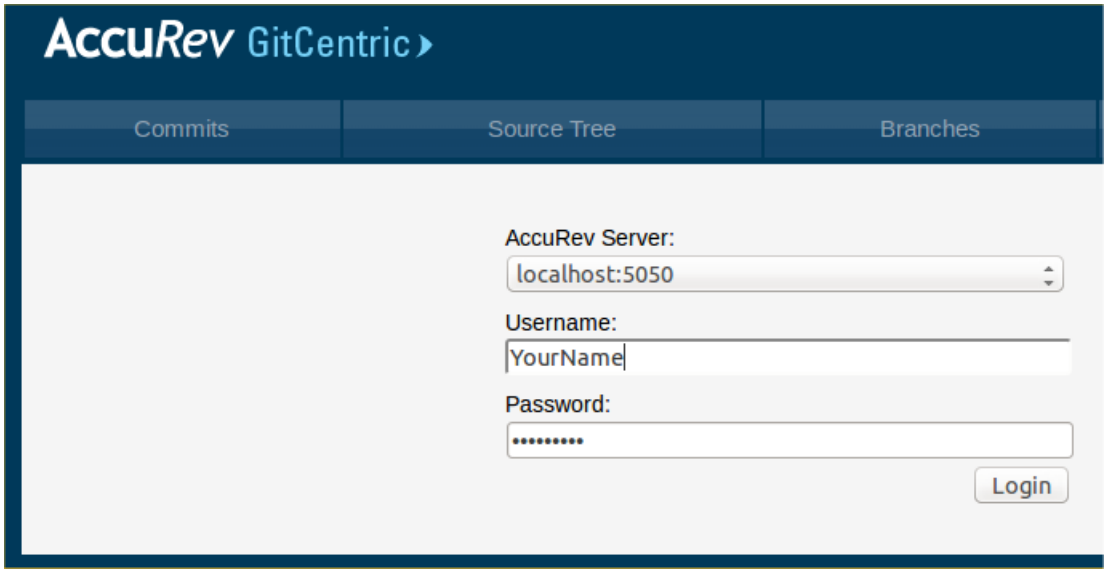
Log In to GitCentric

The first step in using the GitCentric web GUI is logging in with your web browser.

1. Point your browser to the GitCentric web server. For example:

   `http://<servername:port>`

   You will see the following log-in screen and the GitCentric menus will be displayed, although some will be disabled until you log in.



> **Note:** The very first user to log in to GitCentric, who should be the administrator who installed it, will see a slightly different log-in screen, which includes a field for specifying a "CLI Path". This initial login screen and procedure is described in the *GitCentric Installation and Release Notes*.

2. Here are the fields that you may encounter with either of these dialogs. Most are relatively self-explanatory so long as you know that the **Username** and **Password** are your credentials on the AccuRev server.

   AccuRev Server: From the pull-down menu, select the host name of the AccuRev server that you wish to associate with a Git branch, and where you have a login account. For example: `acserver2:5050`, or `<ip_address>:5050` or `localhost:5050`. This may or may not be the same AccuRev server where the GitCentric database resides. In a simple configuration, this is may be the same host that you just connected to with your web browser, or it may be a remote server. Note that if `localhost` appears as an option, it refers to an AccuRev server on the host you are connecting to, not your local machine where you started your browser.

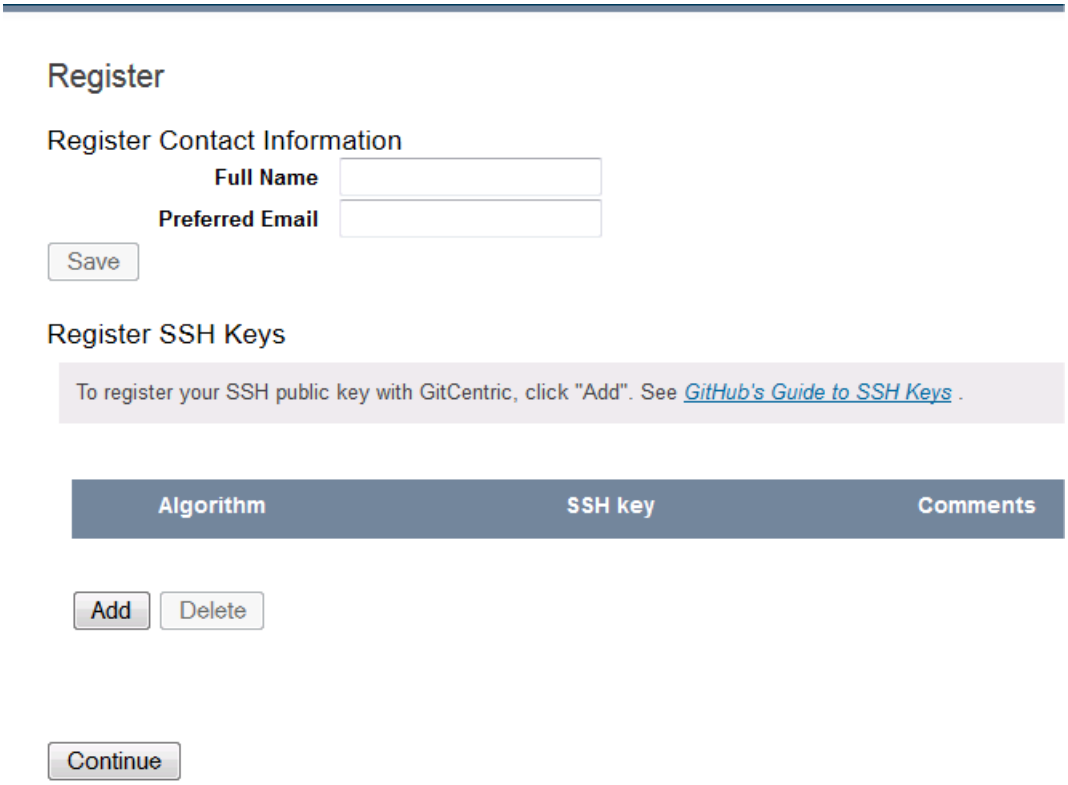
   - **Username:** -- The AccuRev user account on this server that you use. If you are an administrator, this would be the account that you use for GitCentric configuration, and might be an account such as "`acserver`" if you have legacy AccuRev systems. See the *GitCentric Installation and Release Notes* for a discussion about user accounts.
   - **Password:** -- The password for the specified AccuRev user.

3. Click **Login** when done.

If this is your first time logging into GitCentric, you will be taken through the self-registration process described in the next section. Otherwise, you will be taken to the GitCentric UI.

Register with GitCentric

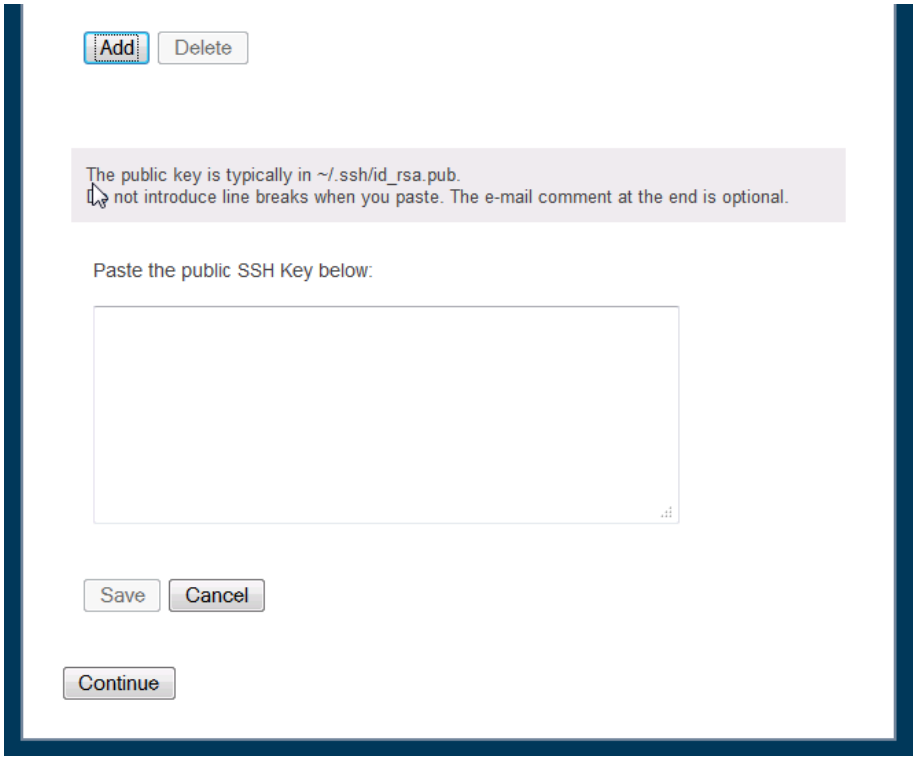
When you log in to GitCentric for the first time, you are prompted through a self-registration process. This is provided so that users with valid AccuRev accounts will not need to involve an administrator in getting set up on GitCentric:



Enter your information as follows:

1. **Full Name:** Your real name (unless this is an account for a role such as “Guest User”.
2. **Register New Email...:** You should enter the same email address here that you specified when you specified your `user.email` configuration for your Git clone(s). You can specify multiple email addresses and switch between them as necessary with the **Preferred email** setting. **Note:** If your AccuRev user account is associated with an email address, that address will be automatically inserted in this field.
3. **Preferred Email:** Enter the same email address here that you specify for the Git `user.email` configuration for your clones. If these do not match, pushes from your clone will fail. If you need to maintain multiple email addresses, you can add them later at **My Account -> Contact Information**. (See *Contact Information* on page 59.)
4. **SSH key:** Click the **Add** button and copy the contents of your SSH public key file here (see *Create an SSH key* on page 14) and **Save** it. (**Note:** If your AccuRev user account is associated with an SSH public key, that key will be automatically inserted in this field.) Your public SSH key file is typically `~/.ssh/id_rsa.pub`. You can choose to do this later through **My Account -> Public Keys** (see *Public*

Keys on page 59), but until this is set, you will be able to use only the code review features of GitCentric.

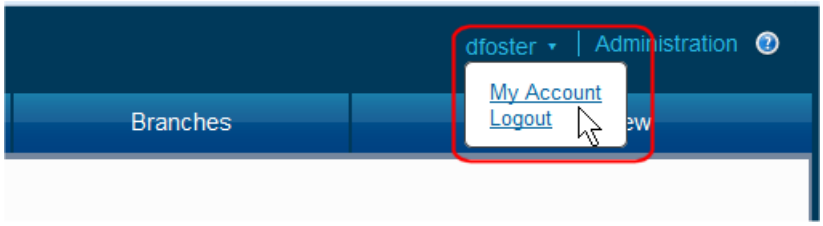


5. Click **Continue** when done.

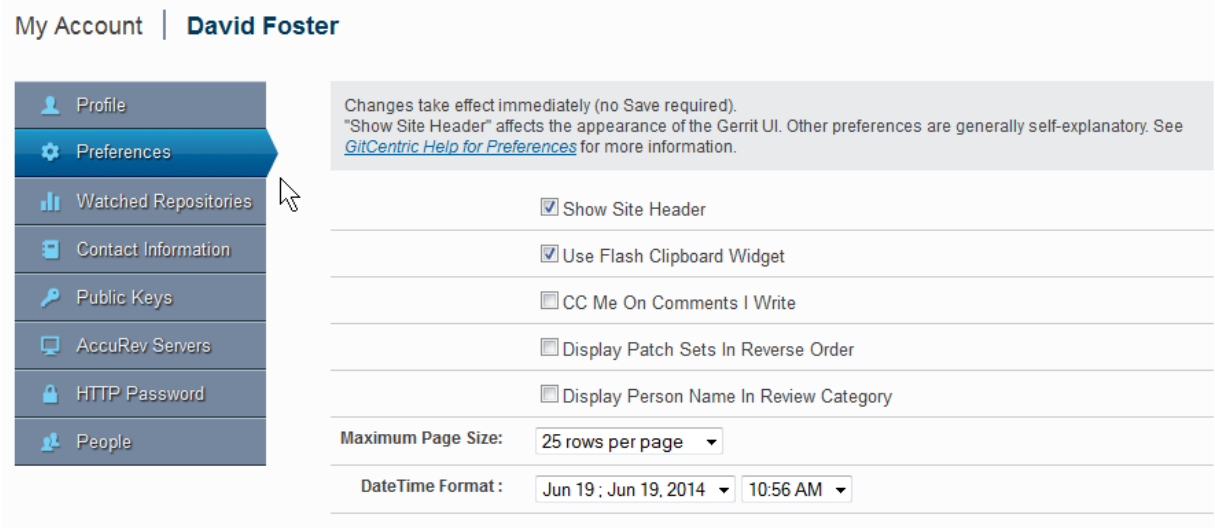
Set Preferences

GitCentric provides a dialog for controlling various aspects of your display.

1. In the upper right corner of the GitCentric window, click **My Account** from the username drop-down.



2. On the My Account page, click **Preferences**.



3. Most of these settings are self-explanatory, and are summarized in *Preferences* on page 58. However, the Show Site Header setting is non-obvious and requires some set-up to make it work. Use the following procedure to set up a site header (and footer) within Gerrit Code Review.

Define and Display a Site Header or Footer for Gerrit Code Review

Gerrit Code Review provides the option of displaying a header and/or a footer in its web UI:

1. Create an image file or files that you want displayed for the header and/or footer. Place the file or files in *<gc_home>*`/site/static`
2. Create an XML-compliant file named `GerritSiteHeader.html` in *<gc_home>*`/site/etc`. Include a pointer to the image file or files in *<gc_home>*`/site/static` that you want to use for the header. Note that this file must be valid XHTML. It is not sufficient for it to be valid HTML. For example, the following example would not work if you left out the superfluous "`</img>`" closing tags. This example creates a site header from two image files displayed horizontally as a table row, with a link to the AccuRev web site from the splash image.

   Example:

```
<div id="gerrit_header">
<table border="0" cellpadding="0" cellspacing="0" id="masthead" width="100%">
<tbody><tr><td valign="center">
```

```
<a href="http://www.accurev.com/">
<img height="150" src="static/splash.jpg"></img></a>
</td>
<td width="100%"></td>
<td align="right" valign="bottom">
<img height="150" src="static/titletop.png"></img>
</td>
</tr>
</tbody>
</table>
</div>
```

3. If you also want a site footer, repeat the previous step for a file named `GerritSiteFooter.html`.
4. To enable the display of the header and footer, in the Web UI click **Settings -> Preferences -> Show Site Header**.

Generate an HTTP Password

You can execute Git commands using SSH or HTTP. If you choose to use HTTP, you will need to provide a generated password. To generate an HTTP password:

1. In the upper right corner of the GitCentric window, click **My Account** from the username drop-down.
2. On the My Account page, click **HTTP Password**.
3. On the HTTP Password page, click the **Generate Password** button.

   The generated password appears in the **Password** field.

   Tip: You can use the clipboard icon ( ) to copy the password.

See *HTTP Password* on page 60 to learn about ways to manage and secure the password when executing Git commands using HTTP.

Create a Clone From a GitCentric Repository

The procedure for creating a local clone from a GitCentric repository is a basic Git operation:

1. `git clone ssh://<gitCentricLogin>@<gitCentricServer>:<port>/<repoName> <cloneName>` where:

   <gitCentricLogin> is the AccuRev username you specified in *Log In to GitCentric* on page 14.

   <gitCentricServer> is the host where GitCentric is installed (the same you specify in the URL in *Log In to GitCentric on page 14*.

   <port> is the SSHD listener port (typically `29418`).

   <repoName> is the name of the GitCentric repo.

   <cloneName> is the name of the clone you are making of GitCentric repo.

   Notes about repo and clone names:

   - Typically *<repoName>* and *<cloneName>* will be the same, but they do not need to be.
   - If the repo has a ".git" extension, you do NOT need to specify it in this command.
   - By convention, only bare repos have a .git extension. Working repos do not. In general, you should NOT specify ".git" extensions when working with GitCentric.

2. Prepare to configure the clone:

   ```
   cd <cloneTopDirectory>
   ```

3. Configure your username:

   ```
   git config user.name "<yourRealName>"
   ```

4. Configure your email address:

   ```
   git config user.email "<yourEmailAddress>"
   ```

   Note: This must be the same as your "Preferred Email" as registered with GitCentric in *Register with GitCentric* on page 16, or else `git push` operations will fail.

Configure the Clone for Code Review (Optional)

If you will be using GitCentric's optional Gerrit Code Review functionality as part your development, you MUST configure your clone as follows:

```
git config remote.origin.push 'refs/heads/*:refs/for/*'
```

This causes files to be pushed to a special branch for code review.

Next, copy the `commit-msg` hook to the `.git/hooks/` directory of each clone. For example:

```
scp -p -P <port><server>:hooks/commit-msg .git/hooks/
```

where:

`<port>` is the SSHD listener port (typically `29418`)

`<server>` the host where GitCentric is installed

If you will not be using Gerrit Code Review, and you are pushing directly to the repository, then leave this step out. In either case, ensure that your GitCentric administrator has set "push" and other ACL permissions correctly for either the Gerrit Code Review or direct push environment. (See *Configure Access Rights (ACLs) for a Repo* on page 34.)

Configure the Clone for Direct Push

If your site does not use the GitCentric's optional Gerrit Code Review functionality, you must ensure that the line `push = refs/heads/*:refs/for/*` (added by the `git config` command described in the previous section) does NOT appear in the `[remote "origin"]` section of your clones' `config` file.

Troubleshoot Git Clone Issues

If you have a problem cloning a repository, or using it once it has been created, check the following:

- Ensure that the user has gone through initial GitCentric login and has the correct username, email, and ssh.
- Test the ssh configuration with this command:

  ```
  ssh -p 29418 <gc_user>@<gitCentricServer> gitcentric --help
  ```

  This should return a usage message that refers to the **config-branch**, **config-repo**, and **ls-repo** commands. If the command returns an error message, ssh has not been set up correctly. In this case, try the command again using a `-v` switch to obtain additional messages that might be useful for debugging.
- Make sure that the path in the `git clone` call is correct.

- If you have an authorization error, have an administrator check the ACLs in **Administration -> Repositories -> *<repoName>* -> Access**.
- Have an administrator check the bare repo in the GitCentric storage directory. Use commands such as `git log`, `git branch`, and `git cat-file -p master`. If it is empty, check the log files for initial import/export failure.

Note: Git does not track empty directories although AccuRev does. Therefore, if you have empty directories in your AccuRev stream, these will NOT appear in a Git clone of a repository that is mapped to that AccuRev stream.

View Commit History

Use the Commits page to view the commit history of a specific branch or tag. You can locate a branch or tag using its name.

To review a branch's or tag's commit history:

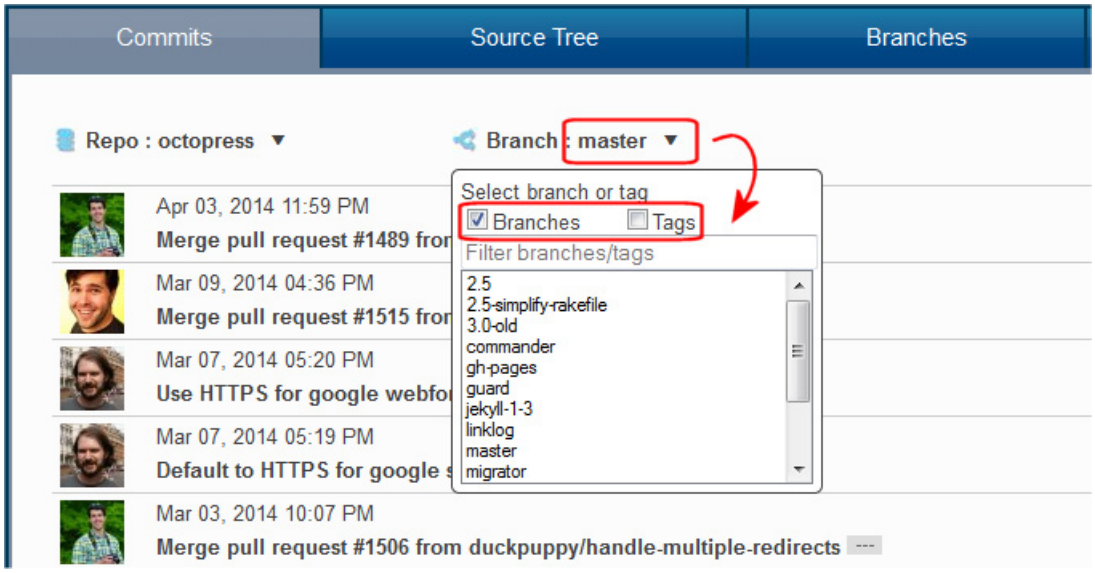
1. Click the **Commits** tab to display the Commits page.

   The **Repo** and **Branch** fields retain their last values, even if they were set on the Source Tree or Branches pages.

2. Select the repository from the **Repo** drop-down menu.

   If available, the master branch for the repository you choose is selected by default in the **Branch** field. Otherwise, GitCentric displays the first branch in the repository based on an alphanumeric sort.

3. Optionally, use the **Branch** drop-down menu to choose a branch other than master or a tag.

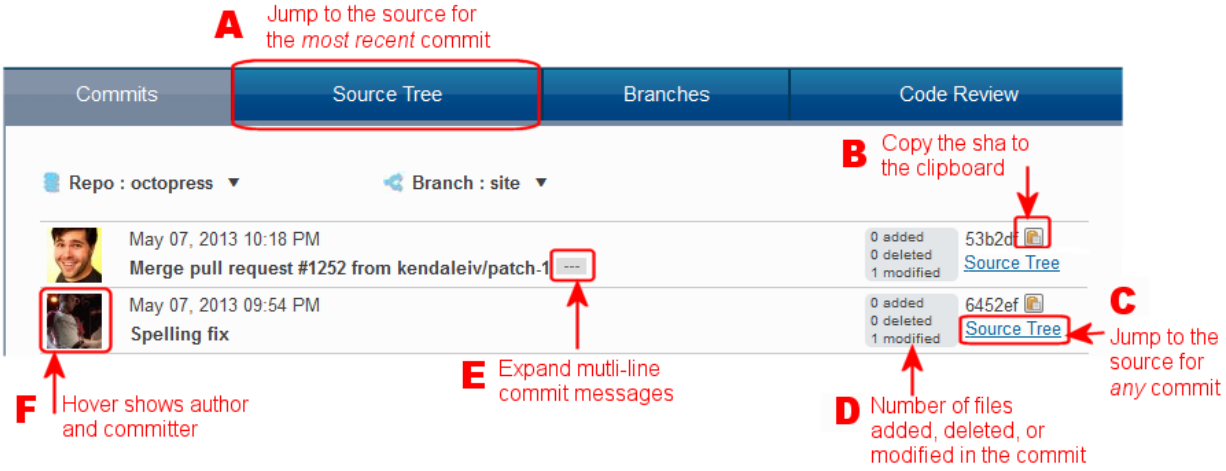


Tip: The list in the drop-down menu includes only branches by default. You can also include (or restrict the list to) *tags*, which are often used to mark important milestones like release points (v2.0, for example). If you choose a tag, GitCentric changes the **Branch** icon and label to **Tag**: Tag : .

GitCentric refreshes the Commits page to update the starting point of the history. Each row represents a single commit, starting with the most recent. Scroll the page to reach the origin of the commit graph.

Information Displayed on the Commits Page

As shown in this example, the Commits page provides several features that make it easy to view and explore commit history:



Item	Feature	Description
A	**Source Tree** tab	Jumps to the Source Tree page for the most recent commit in the branch. See *View Commit History* for more information. Note that you can change the repository and branch using the **Repo** and **Branch** drop-down.
B	Clipboard	Copies the commit's sha to the clipboard -- this allows for easy use of the sha in a Git command, for example.
C	`Source Tree` link	Jumps to the Source Tree page for any commit you choose. See *Review a Commit's Files* for more information.
D	Commit summary	Shows the total number of files that were added to, deleted from, or modified in the commit. Together, this information provides you with a general feel for the size of the commit.
E	More button	Expands the commit row to display the entire message if the commit message consists of multiple lines, or is too long to be displayed in the available space. Click again to return to the default display.
F	Gravatar	Displays the names of the commit author and committer when the pointer is placed over the *gravatar* (Globally Recognized Avatar). If the author has registered with *gravatar.com*, his or her gravatar appears here; otherwise, a generic image is used.

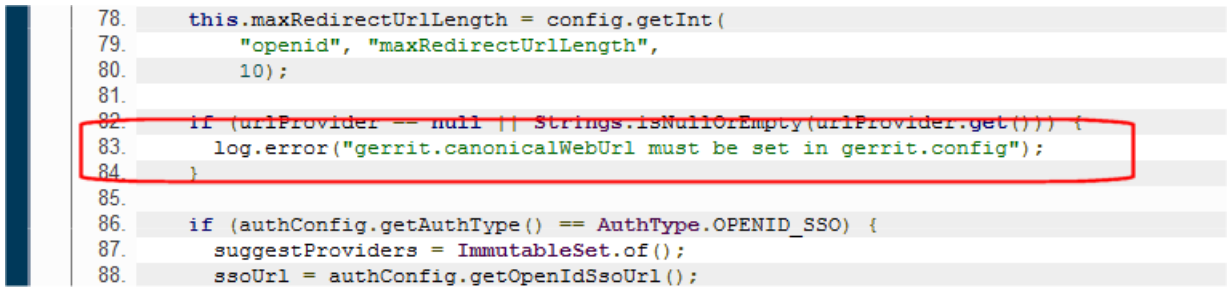
Review a Commit's Files

Use the Source Tree page to review the files -- new, deleted, or modified -- associated with a specific commit. Clicking a text file displays its contents in a file viewer. The contents of binary files (graphics and executables, for example) cannot be displayed.

The ability to view file contents can help developers troubleshoot problems. For example, imagine that a customer using version 6.2.1 of your software reports an error, referencing the following log message:

```
2014-07-07 17:27:47,858 ERROR LoginForm:83 gerrit.canonicalWebUrl must be set in
gerrit.config
```

You could locate the v6.2.1 tag in the Source Tree page, navigate the directory path, and open **LoginForm.java** to view line 83.



The Source Tree page represents a snapshot of the repository at the time of the selected commit. Each row on the Source Tree page represents a directory or file in the source code. You can start from either the Commits or Source Tree page as described in the following sections.

Starting from the Commits Page

To review the files in a commit from the Commits page:

1. Click the **Commits** tab to display the Commits page.

   The **Repo** and **Branch** fields retain their last values, even if they were set on the Source Tree or Branches pages.

2. Select the repository from the **Repo** drop-down menu.

   If available, the master branch for the repository you choose is selected by default in the **Branch** field. Otherwise, GitCentric displays the first branch in the repository based on an alphanumeric sort.

3. Optionally, use the **Branch** drop-down menu to choose a branch other than Master or a tag.

   Tip: The list in the drop-down menu includes only branches by default. You can also include (or restrict the list to) *tags*, which are often used to mark important milestones like release points (v2.0, for example). If you choose a tag, GitCentric changes the **Branch** icon and label to **Tag**: Tag : .

   GitCentric refreshes the Commits page to update the starting point of the history. Each row represents a single commit, starting with the most recent. Scroll the page to reach the origin of the commit graph.

4. Locate the commit whose files you want to review and click the `Source Tree` link.

   Tip: If you want to review the files for the first commit in the current branch or tag, click the **Source Tree** tab.

The Source Tree page appears. See *Information Displayed on the Source Tree Page* for more information.

5. Click the folders associated with the gravatar of the user whose commit you are reviewing. GitCentric updates the path displayed in **Current Directory** field as you navigate the source tree.

   Tip: The name of this field changes to **Current File** when you select a file for viewing.

Starting from the Source Tree Page

To review the files in a commit from the Source Tree page:

1. Click the **Source Tree** tab to display the Source Tree page.

   The **Repo** and **Branch** fields retain their last values, even if they were set on the Commits or Branches pages.

2. Select the repository from the **Repo** drop-down menu.

   If available, the master branch for the repository you choose is selected by default in the **Branch** field. Otherwise, GitCentric displays the first branch in the repository based on an alphanumeric sort.

3. Optionally, use the **Branch** drop-down menu to choose a branch other than Master or a tag.

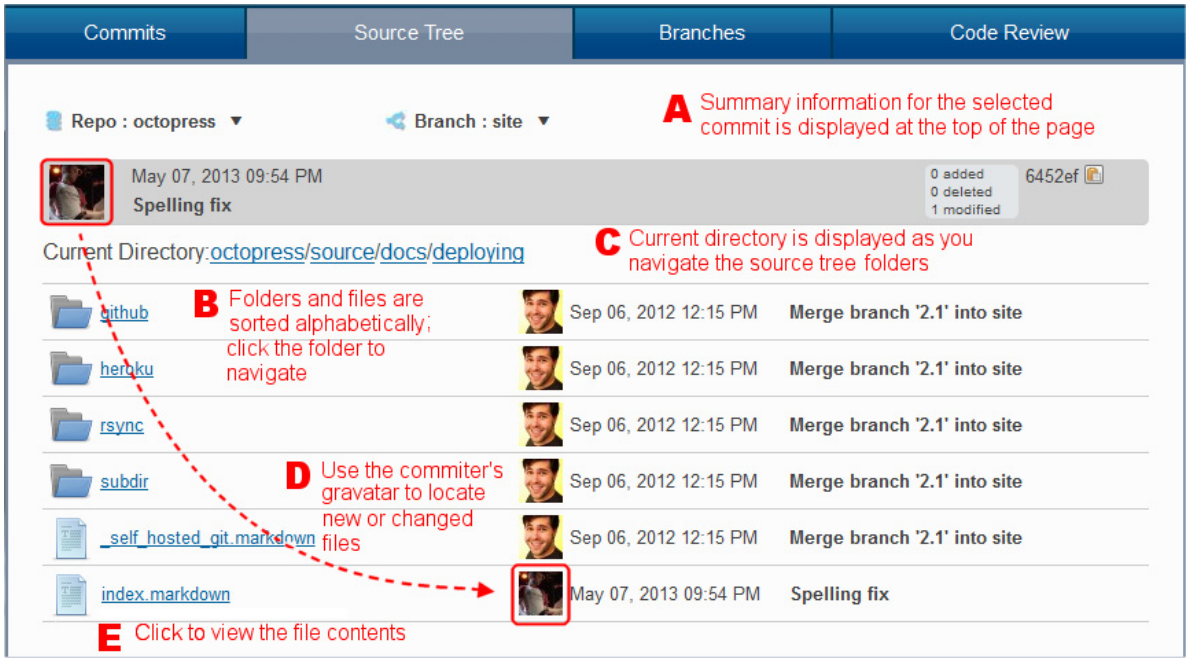
   Tip: The list includes only branches by default. You can also include (or restrict the list to) *tags*, which are often used to mark important milestones like release points (v2.0, for example).

4. Click the folders associated with the gravatar of the user whose commit you are reviewing. GitCentric updates the path displayed in **Current Directory** field as you navigate the source tree.

   Tip: The name of this field changes to **Current File** when you select a file for viewing.

Information Displayed on the Source Tree Page

As shown in this example, the Source Tree page provides several features that make it easy to locate new, deleted, and modified files in a commit:



Item	Feature	Description
A	Commit summary	The commit summary of the commit you selected on the Commits page is displayed at the top of the Source Tree page to provide context for the displayed source folders and files. It retains the features from the Commits page. See *Information Displayed on the Commits Page* for more information.
B	Navigation	Once on the Source Tree page, you navigate the tree by clicking folders you want to explore. Folders and their files are sorted in ascending alphabetical order. See *Gravatars* for navigation tips.
C	Source tree path	As you navigate the source tree, GitCentric displays the full path of the current directory.
D	Gravatars	You can use the author's gravatar to quickly locate all files and directories that he or she was the last one to touch. The date next to the gravatar tells how recent the change was. If the author has registered with *gravatar.com*, his or her gravatar appears here; otherwise, a generic image is used.
E	File viewer	GitCentric displays text files in a viewer that replaces the main panel of the Source Tree page when you click the file. The contents of binary files (graphics and executables, for example) cannot be displayed. To return to the previous view, you can use the browser's back button, or you can click the desired directory in the **Current File** field.

Compare Branches

During development, branches diverge from other branches in your repository -- you might have committed changes in one branch that have not been merged into another, and vice versa. This type of change is normal and to be expected, but over time it can lead to the creation of "dead" or "stale" branches that can bloat your repository and make development difficult. You can use the Branches page to determine how divergent the branches in the repository are and then take steps to address it (by merging or pruning, for example).

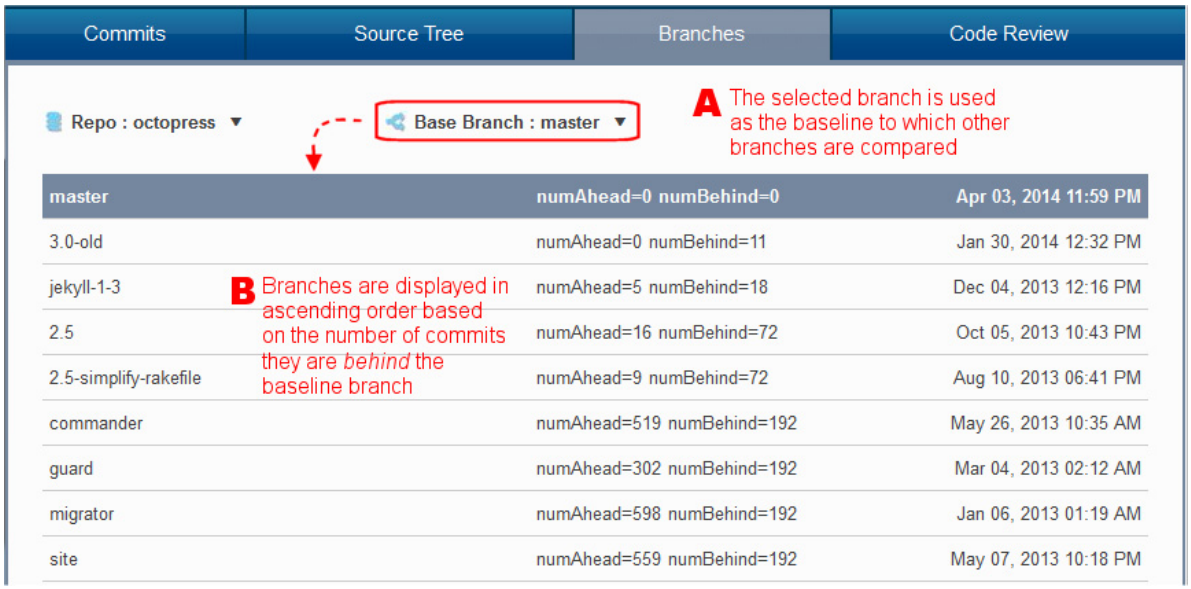
GitCentric presents information about a branch's divergence in terms of how far *ahead* (numAhead) and how far *behind* (numBehind) the commits in a given branch are from those in the branch you choose as your baseline. Consider the following illustration of a simple commit graph, where each **O** represents a commit:

```
---O---O---O---O---O----------O (master)
    \
     ---O---O---O (5.1)
```

Here, branch 5.1 is considered to be 3 commits ahead of master, and 5 commits behind master.

To compare one branch with others in the repository:

1. Click the **Branches** tab to display the Branches page.



   The **Repo** and **Base Branch** fields retain their last values, even if they were set on the Commits or Source Tree pages.

2. If necessary, select the repository from the **Repo** drop-down menu.

   If available, the master branch for the repository you choose is selected by default in the **Branch** field. Otherwise, GitCentric displays the first branch in the repository based on an alphanumeric sort.

3. Use the **Base Branch** drop-down menu to choose the branch against which you wish to compare all other branches in the repository. For example, if you want to see how a the site branch compares to master, you would choose master.

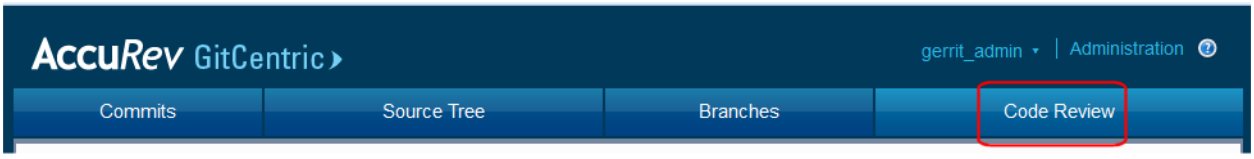
   Note: You cannot compare tags using the Branches page.

   The branch you choose as the baseline is moved to the top of the page and highlighted. The other branches in the repository are displayed in ascending ordered by the number of commits they are behind the baseline branch (numBehind).

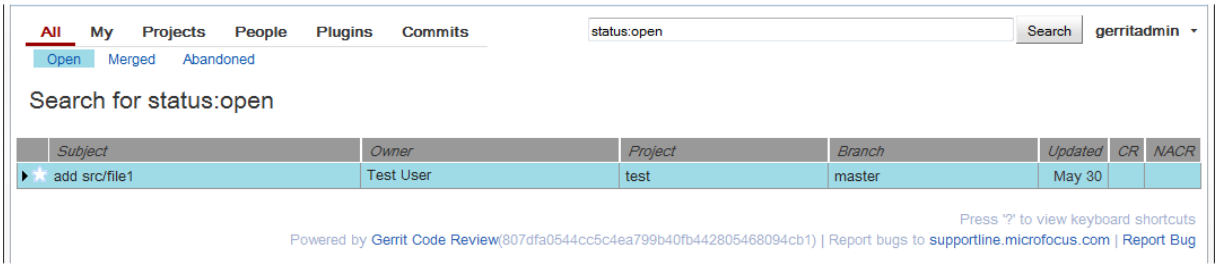
Switch Between Gerrit Code Review and GitCentric

GitCentric incorporates the open-source Gerrit Code Review tool. Your site may or may not have made code review mandatory. If it is mandatory, then when you push your changes in Git, they must be approved by other reviewers before they are accepted into the repository. You may be asked to be a reviewer for other people's changes. If code review is not mandatory at your site, then when you push your changes, they go directly into the repository.

To use GitCentric's optional Gerrit Code Review feature, click **Code Review** at the top of the GitCentric UI.



Doing so displays the Gerrit Code Review interface in the same browser window previously occupied by the GitCentric UI.



There are a number of ways to return to the GitCentric UI:

In Code Review, Clicking	Brings You To
The browser's back button	The GitCentric page you were on prior to opening Gerrit Code Review
Projects	The Repositories page (in GitCentric: **Administration** > **Repositories**)
People	The Groups page (in GitCentric: **Administration** > **People**)
Commits	The Commits page (in GitCentric: **Commits** tab)
username > **Settings**	The My Account page (in GitCentric: username > **My Account**)

Gerrit Code Review has its own documentation, which you can find here:

http://gerrit-documentation.googlecode.com/svn/Documentation/2.7/index.html

Tip: Experienced users of Gerrit Code Review should note that all administrator and user profile functions are accessed through the GitCentric UI.

Procedures for Administrators Only

The procedures in this section apply to only members of the Administrators group. Users that do not belong to this group cannot access this functionality, and cannot view items to which they do not have permissions.

Table 2. Summary of Common GitCentric Procedures for Administrators

To:	Go to page:
Configure AccuRev	*28*
Create a Repository for GitCentric	*29*
Remove a Branch or a Repository	*31*
Remove a Repository	*31*
Import an Existing Git Repo	*31*
Set General Attributes for a Repo	*33*
Create Branches for a Repo	*34*
Configure Access Rights (ACLs) for a Repo	*34*
Map a Git Branch to an AccuRev Stream	*38*
Unmap a Git Branch from an AccuRev Stream	*42*
Enable and Use Change Packages	*43*
Add an AccuRev Server	*46*
Manage GitCentric Groups	*49*
Configure AccuRev Element ACLs (EACLs)	*51*
Enable/Disable Code Review	*52*
Enable Gerrit Code Review Replication	*53*
Troubleshoot Import/Export Operations	*54*
Set Up Gerrit Garbage Collection	*55*

Configure AccuRev

1. On the AccuRev server, make sure that "server_master_trig.pl" has been installed in the directory `<ac_home>/storage/site_slice/triggers`

   See the *GitCentric Installation and Release Notes* for details. (You will need to do this for each AccuRev server that works with GitCentric. For now, we assume that you are configuring just your first AccuRev server.)

2. Configure a stream to which your Git repository branch will map. Some guidelines:

   - Identify the stream where you will be sharing files between AccuRev and Git. Determine which directory in this stream will be the mount point.
   - Alternatively, if you need include/exclude rules to control which elements are visible to GitCentric, create a pass-through stream underneath the target stream, configure your include/

exclude rules on the pass-through stream, and map to that instead. (For best performance, if the target stream contains cross-links, create a pass-through stream and exclude the cross-links.)

- You cannot use the root stream of a depot for GitCentric, so if you are targeting the root, you *must* create a sub-stream to map.
- If you will be using AccuRev element ACLs to control access to AccuRev-controlled files, make sure that your users, groups, and EACLs are defined and applied the way you want them. For example, you may plan to have one Git repository available to an off-shore team that will have only limited access to your files, while another Git repo is used by your own, domestic development team that will expect to have access to all Engineering files. In this case, you would want to set up your EACLs in AccuRev so that the off-shore user group will see only the files they need. See the EACL documentation in the AccuRev *Administrator's Guide* for information about configuring this aspect of security.

Configure GitCentric

Once your AccuRev server is configured, use the procedures in this chapter to create and configure your Git repositories:

1. Log in to GitCentric (see *Log In to GitCentric* on page 14).
2. Add the AccuRev server to the GitCentric database (see *Add an AccuRev Server* on page 46).
3. Add a repository using GitCentric and map it to an AccuRev stream (see *Create a Repository for GitCentric* on page 29).
4. Set the security for the repository (see *Access* on page 68).

Create a Repository for GitCentric

A common task you will perform as a GitCentric administrator is creating a new Git repository for use with GitCentric, and optionally, mapping its branches to AccuRev streams. (If you have an existing repo that you wish to register with GitCentric, proceed to *Import an Existing Git Repo* on page 31.)

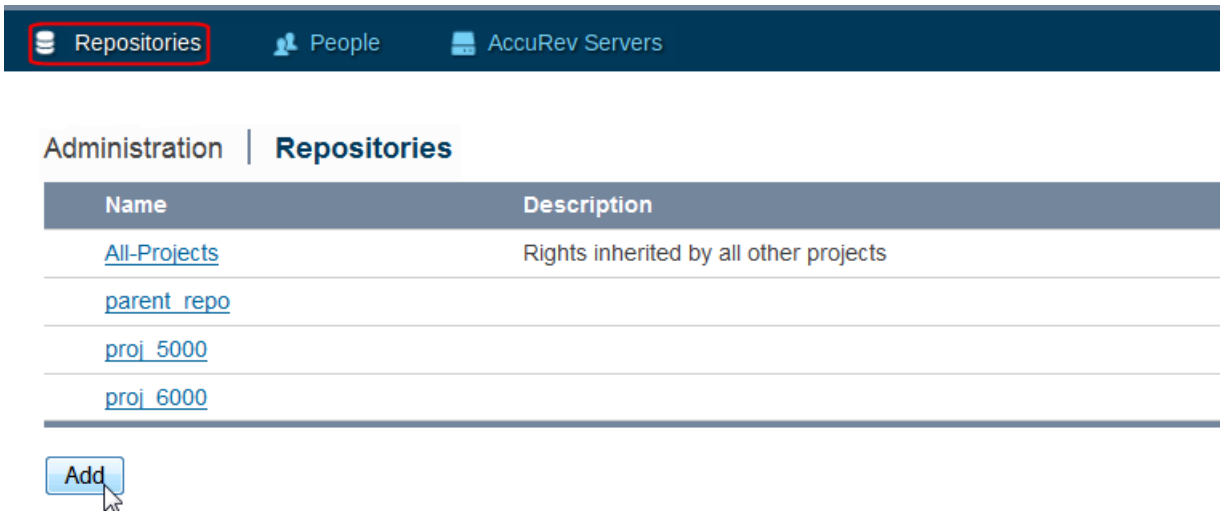
The following procedures assume that:

- You have installed GitCentric as described in the AccuRev *GitCentric Installation and Release Notes*.
- You have read through *Chapter 1 Concepts and Overview*, particularly *GitCentric Administrators and Mapping Branches to Streams* on page 4, *Mapped Behavior* on page 5, and *Best Practices When Planning Your Installation* on page 5.
- The AccuRev server that was added to GitCentric during the initial login process is the server that contains a stream that you want to map to a Git branch. If you need to make another AccuRev server known to GitCentric before mapping a Git branch, proceed to *Add an AccuRev Server* on page 46, and then return here when done.

To create a new repo:

1. Log in to GitCentric as described in *Log In to GitCentric* on page 14.

2. When the GitCentric GUI appears, click the **Administration** button, then click the **Repositories** menu to display the Repositories page. (See *Chapter 5 Administration* for reference information.)



3. Click the **Add** button and populate the fields in the resulting panel.

- **Repository Name**: Do not specify a ".git" extension for the repository name. GitCentric adds them on creation.
- **Inherit Rights From**: By default, new repos inherit their rights from the system-defined "All Projects".
- **Only serve as parent for other projects**: Creating a parent-only project means that you can set ACLs on it and they get inherited by all its children. This also makes it easier to change the configuration of a number of repos at one time using the `--children-of` option to the `gitcentric config-repo` and `config-branch` CLI commands (see *Appendix C Command-Line Reference*).

4. Click **Save** when done.

Once a repository is created, you can set general attributes on it, create branches, configure access rights, and map its branches to AccuRev streams, as described in the following sections.

Remove a Branch or a Repository

These operations require manual procedures as described below.

Remove a Branch

There is no GitCentric "Delete Branch" feature. To remove a branch that has been mapped to an AccuRev stream, use this procedure:

1. Ensure that the **Force** option is applied to the **Push** and **Push Annotated Tag** permissions for the repo (see *Access* on page 68).
2. Use the GitCentric config-branch CLI command (see *config-branch* on page 90) to clear any GitCentric mappings:

   ```
   ssh -p 29418 <username>@<server> gitcentric config-branch
        --branch <branchname> <reponame> --clear
   ```

   Alternative: On the AccuRev Connector panel (**Administration** > **Repositories** > **AccuRev Connector**), select the mappings associated with the branch you intend to delete and click the **Delete** button.
3. Use the standard Git command to delete the branch:

   ```
   git branch -d $branch
   ```
4. Push the change.

Remove a Repository

There is no "Delete Repository" feature in the GitCentric UI. You should not remove a repository from disk. Instead, set the description to "OBSOLETE" and set the ACLs so that only the Administrator group can see it.

If you are certain you want to remove a repository, consider using the `gitcentric delete-repo`. See *delete-repo* on page 94 in *Appendix C Command-Line Reference* for more information.

Import an Existing Git Repo

If you have an existing Git repository that you would like to register with GitCentric and import into AccuRev, you can import a snapshot of the current state of the repository into AccuRev.

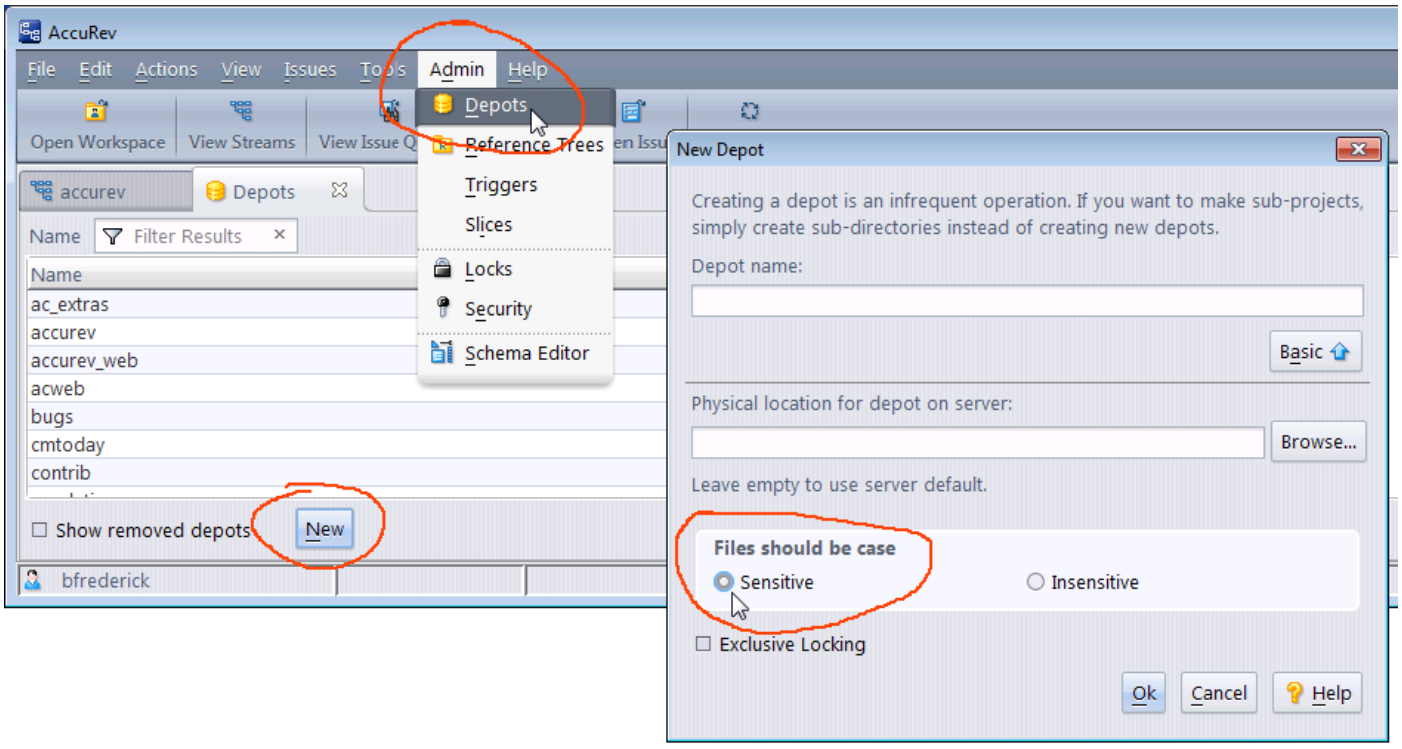
A Note About AccuRev Depots

You must import a Git repository into a *case-sensitive* AccuRev depot, unless you are 100% certain that repo is case-insensitive.

Your Git repo will be case-sensitive if it has been created and developed exclusively in a Linux/UNIX environment. If it has included any development from Windows or OS X developers, there is a good chance (but not 100% certainty) that the repo is case-insensitive, even if the repo itself is hosted on a Linux/UNIX server. If you attempt to map a case-sensitive repo to a case-insensitive depot, you will encounter errors, either at import time, or further down the road.

By default, AccuRev depots are case-insensitive, and the depot that is created during a new AccuRev installation is always case-insensitive. If you need a case-sensitive AccuRev depot for use with GitCentric, you (or an AccuRev administrator) must create one using AccuRev depot GUI features shown in the

following illustration. See the AccuRev documentation for more information, or for the equivalent CLI commands to perform the same operation.



Import a Snapshot of the Latest Heads into AccuRev

If you do not care about capturing the repo history, the process can be as simple as making a copy of the repository under the GitCentric repo storage directory. The next time you restart the GitCentric server the repo will be visible through the GitCentric UI.

To import a repository and just capture its current state:

1. Create a copy of the repo in your GitCentric repository home.

   `cd <gc_repo_home>` (For example, `/home/<gc_installer>/AccuRevGitCentric/site/git`)

   `cp -r <orig_repo>/<new_repo_name>`

2. Restart the GitCentric server. You have two options to accomplish this:

   - Stopping and restarting the Tomcat web server:

     ```
     cd <ac_home>/WebUI/tomcat/bin
     ./shutdown.sh
     ./startup.sh
     ```

     Note: Make sure that the user starting Tomcat has write access to the `logs`, `temp`, `webapps`, and `work` directories in `<ac_home>/WebUI/tomcat`. This user should have read access to all other Tomcat directories and files.

   - Flushing the cache with an SSH command (entered on a single line):

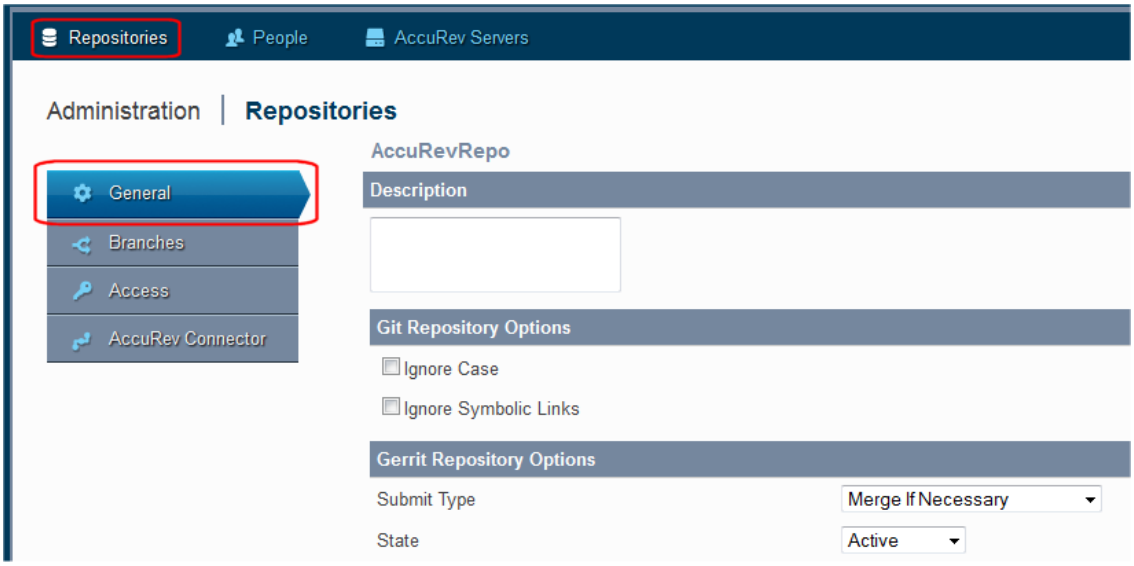
     ```
     ssh -p 29418 <username>@<gc_server> gerrit flush-caches --cache projects
           --cache project_list
     ```

3. Configure the repository through GitCentric and allow users to clone it as necessary as described in the other sections of this chapter.

Set General Attributes for a Repo

1. Click the **Administration** button, then click the **Repositories** menu to display the Repositories page. (See *Chapter 5 Administration* for reference information.)
2. Click the repository whose attributes you want to set:

This brings up the **General** page, where you can set various attributes for the currently selected repository.



For a description of the various options, see the reference page for this display, at *General* on page 66.

Create Branches for a Repo

1. Click the **Administration** button, then click the **Repositories** menu to display the Repositories page. (See *Chapter 5 Administration* for reference information.)
2. On the Repositories page, click the **Branches** menu.
3. Click **Create Branch**. The fields are pre-populated with prompts to assist you with branch creation (see the reference page for this at *Branches* on page 67 for more information).

Configure Access Rights (ACLs) for a Repo

GitCentric implements a group-based ACLs security model derived from Gerrit Code Review. This is a powerful tool for defining who can do what with Git repositories under GitCentric control. You will need to have a solid understanding of these group-based ACLs and do some planning before implementing them, so be sure to first read the following section: *GitCentric Group-Based ACLs* on page 10, and the *Gerrit Code Review* documentation for details.

General Procedure for Setting ACLs

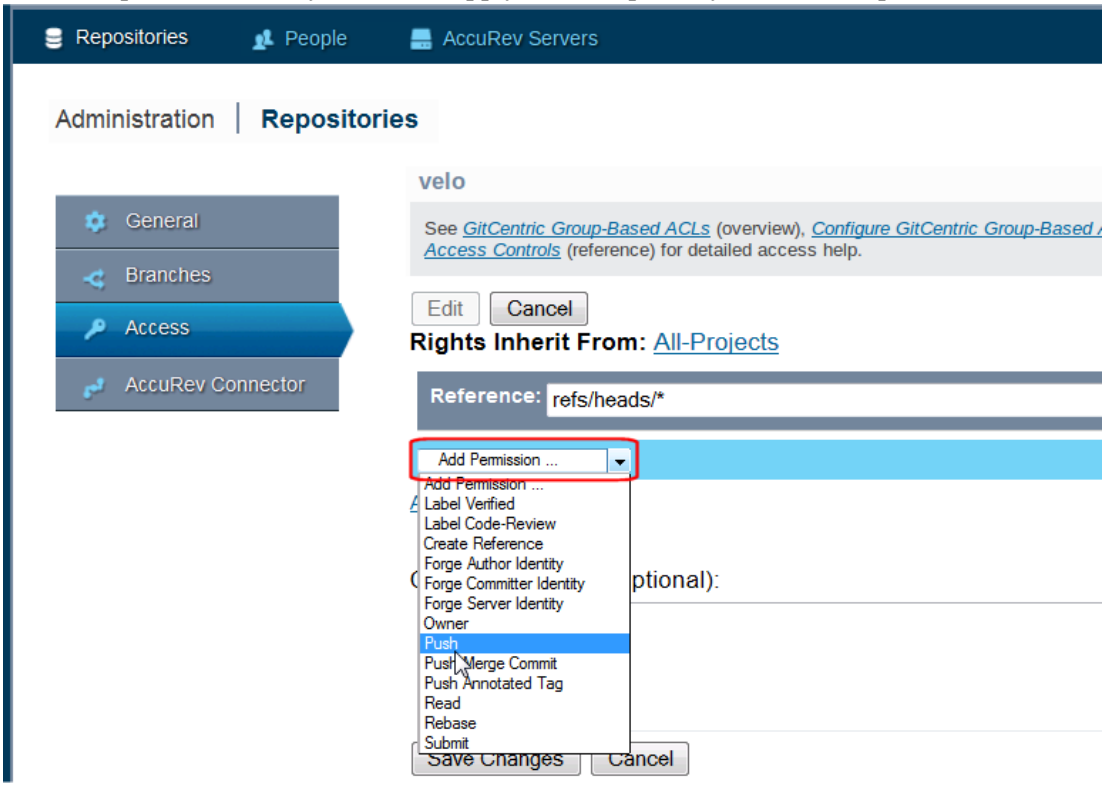
Use the following general procedure for settings ACLs on your repos.

1. Click the **Administration** button, then click the **Repositories** menu to display the Repositories page. (See *Chapter 5 Administration* for reference information.)
2. Click the repository you want to configure. Selecting All-Projects will cause all other repositories to inherit whatever ACLs you apply. (**Note:** Only the All-Projects system-defined repo has a "Global Capabilities" section with an **Administrate Server** permission which gives Administrators broad powers across all repos. Be extremely careful when editing this permission.)

3. On the Repositories page, click the **Access** button. (Reference information is a available at *Access* on page 68.)

4. Use the **Rights Inherit From:** link to specify the repository from which you want the new repository to inherit its access settings. (By default, all repositories inherit access rights from the system-defined "All-Projects". You can create a parent-only repository to easily apply settings such as this to all child repositories. See *Create a Repository for GitCentric* on page 29 for more information.)
5. Click **Edit**, then **Add Reference**. In the **Reference** field, you can accept the default value (`refs/heads/*`) to apply the access setting to all branches in repository, or modify it to apply to a specific branch. Specifying `refs/*` applies the ACL to everything in the repo.

6. Select the permission that you want to apply to this repository from the drop-down menu.



7. After selecting the permission, complete the Group Name field for whom this permission applies in the this repository. The field supports type-ahead, so, for instance, entering “R” will provide “Registered Users” as a possible completion.

8. Optionally enter a commit message. An admin user can view this when performing a `git log refs/meta/config` command on the repository.

9. When done, click the **Save Changes** button.

Configuring ACLs for Code Review

Part of enabling the GitCentric’s optional Gerrit Code Review functionality involves setting ACLs for Git references as shown in the following table:

Table 3. ACL Settings Required to Support Code Review

Reference	Permission	Setting	Group
`refs/*`	**Forge Author Identity** **Submit**	**ALLOW**	**Registered Users**

Table 3. ACL Settings Required to Support Code Review

Reference	Permission	Setting	Group
refs/for/refs/*	**Push**	**ALLOW**	**Registered Users**
refs/heads/*	**Label Code-Review (range -2 to +2)**	**ALLOW**	**Registered Users**

Most of these ACLs are set automatically for the All-Projects repository when you install GitCentric. There are slight differences based on whether you installed GitCentric 2013.3 for the first time, or you upgraded to GitCentric 2013.3 from an existing installation. If you:

- *Installed GitCentric 2013.3 for the first time*, all ACLs listed in the preceding table are set as the default for the system-defined "All-Projects" repository. Any repository that inherits rights from "All-Projects" is also configured to support code review.
- *Upgraded to GitCentric 2013.3 from a previous release* and you are currently using Gerrit Code Review, you will need to manually set the Submit permission for refs/* as shown in the preceding table. All other permissions (Forge Author Identity, Push, and Label Code-Review) were set as the default for All-Projects when you installed GitCentric.

  If you upgraded from a previous release and are *not* currently using Gerrit Code Review and now wish to, you must manually set all the ACLs listed in the preceding table.

  See *General Procedure for Setting ACLs* on page 34 for more information.

Once the ACLs are set, all that remains to enable code review is for your users to configure their clones to support code review. See *Configure the Clone for Code Review (Optional)* on page 20 for more information.

Additional Considerations for ACLs

There is no one set of ACLs that are appropriate for every installation; you must analyze the needs of your specific installation and adjust the default ACLs as necessary. See the *Access Controls topic* in the Gerrit Code Review documentation, specifically the section labeled "Examples of typical roles in a project" for suggestions about setting Gerrit Code Review ACLs.

The remaining topics in this section describe other considerations to evaluate when enabling code review.

Label Verified

The "Verified" category is generally intended to be used in continuous integration environments where Gerrit Code Review is integrated with Hudson or Jenkins. Typically, the vote for this category is set to +1 by the integration tool indicating that it was able to compile and run tests on the change, allowing the change to be submitted. If you do not have a continuous integration environment, you can either disable the "Verified" category, or configure the "Label Verified" ACL in GitCentric to allow users to manually specify a vote via the Gerrit Code Review GUI. The ACL to allow manual voting for the "Verified" category is shown in the following table.

Table 4. Setting Label Verified for Code Review

Reference	Permission	Setting	Group
refs/heads/*	**Label Verified (range -1 to +1)**	**ALLOW**	**Registered Users**

Note: Specifying the "Label Verified" ACL this way permits any Registered User to approve and submit any code review change. This may be fine for testing, but you must tighten up ACLs for production.

Allowing users to set "Label Verified" manually could be redundant with the "Label Code-Review" ACL. These three approaches to configuring the "Verified" category are summarized below in the order of desirability:

1. Configure Verify with Hudson or Jenkins for automated, Continuous Integration (CI).
2. Disable Verify completely.
3. Configure the "Label Verified" ACL in GitCentric (as shown in the preceding table) to allow users to vote manually in the Gerrit Code Review UI.

If you choose the third option for testing purposes, you should consider either reconfiguring it for automated CI, or disabling it completely, prior to putting GitCentric into production mode.

Configuring "Verify" for Continuous Integration

With Hudson or Jenkins, you would typically set the Label Verified permission for the "Non-interactive users" group. See the Hudson, Jenkins, and/or Gerrit Code Review documentation for details about configuring a Continuous Integration environment.

Disabling "Verify"

If you are not using Jenkins/Hudson continuous integration, you can disable the "Verified" requirement in Gerrit Code Review with the `gerrit gsql` command:

```
ssh -p 29418 <your_server.com> gerrit gsql
DELETE FROM approval_categories      WHERE category_id = 'VRIF';
DELETE FROM approval_category_values WHERE category_id = 'VRIF';
```

Configuring GitCentric ACLs for Direct Push

If you do not wish to use GitCentric's optional Gerrit Code Review functionality, you must modify the default ACLs as follows. (These are simply minimal suggestions. You must analyze your site requirements and customize these settings as appropriate for your specific site.)

1. Change the setting for the "Push" permission for `refs/for/refs/*` to BLOCK.
2. Add the "Push" permission for `refs/*` and set it to ALLOW.
3. Add the "Push Merge Commits" permission to `refs/*` and set it to ALLOW.

You must also have your users adjust the Git `config` file in each of their clones to remove the following line:

```
push = refs/heads/*:refs/for/*
```

Map a Git Branch to an AccuRev Stream

You associate Git repositories and AccuRev depots at the branch and stream level, respectively: you map a specific branch in a Git repo to a "mount point" (a directory) within a specific AccuRev stream. This process has two main steps, each of which is performed on the AccuRev Connector panel in the GitCentric GUI:

1. Specify the AccuRev server connection.
2. Create the branch-stream mappings.

You also use the AccuRev Connector panel to implement AccuRev *change packages*, which require that all changes be associated with a specific issue from your issue tracking system. See *Enable and Use Change Packages* on page 43 for more information.

Avoid Git Reserved Name for AccuRev Elements

You should avoid using the Git reserved name ".git" for any element (a directory, file, or link, for example) in an AccuRev stream that is mapped to a Git repository. Using Git reserved words like ".git" for AccuRev elements creates problems when GitCentric synchronizes the stream and repository.

Specifying the AccuRev Server Connection

1. Click the **Administration** button, then click the **Repositories** menu to display the Repositories page. (See *Chapter 5 Administration* for reference information.)
2. Click the repository you want to associate with AccuRev.
3. On the Repositories page, click **AccuRev Connector** menu. (Reference information is a available at *AccuRev Connector* on page 68.)

   It is important that you specify the correct account here, so read carefully. The “Service Account” is the “robot” AccuRev user account that GitCentric uses to keep Git and AccuRev in sync. You may have different Service Accounts defined for different AccuRev servers and Git repositories that are under GitCentric control.

   - **Server name:port:** Specify the host name (or IP address) of the AccuRev server that you wish to associate with this repository. If this is a new installation, and your GitCentric AccuRev server also functions as a regular AccuRev server, then you may have only one choice here. Other AccuRev servers can be added via the Servers tab (see *Add an AccuRev Server* on page 46).
   - **Username:** Specify the special AccuRev user that you defined during installation to perform GitCentric automated operations. This should be an account that is a member of the group that is assigned to the “ASSIGN_USER_PRIVILEGE” setting in the acserver.cnf file of the AccuRev server being mapped to this GitCentric branch. By convention, this account is often “gcSyncUser”, but it may be something different at your site. (See the notes under *Basic Architecture* on page 3 for more information.)
   - **Password:** Specify the password associated with the account specified in the previous field.
4. Click the **Save** button to test the AccuRev server connection.

What to Do Next

Once you have created a valid AccuRev server connection, you can create your branch-stream mappings for the repository as described in the following section, *Mapping the Branch to the Stream*.

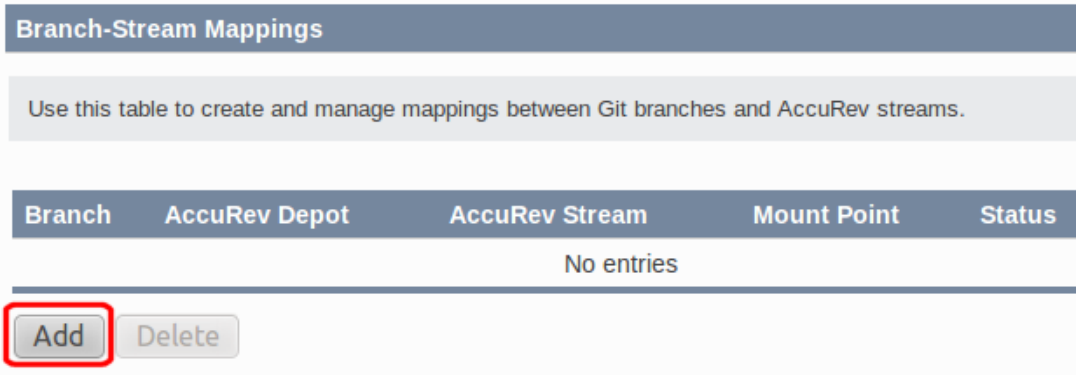
You can also take this opportunity to implement AccuRev change packages. This procedure is described in *Enable and Use Change Packages* on page 43.

Mapping the Branch to the Stream

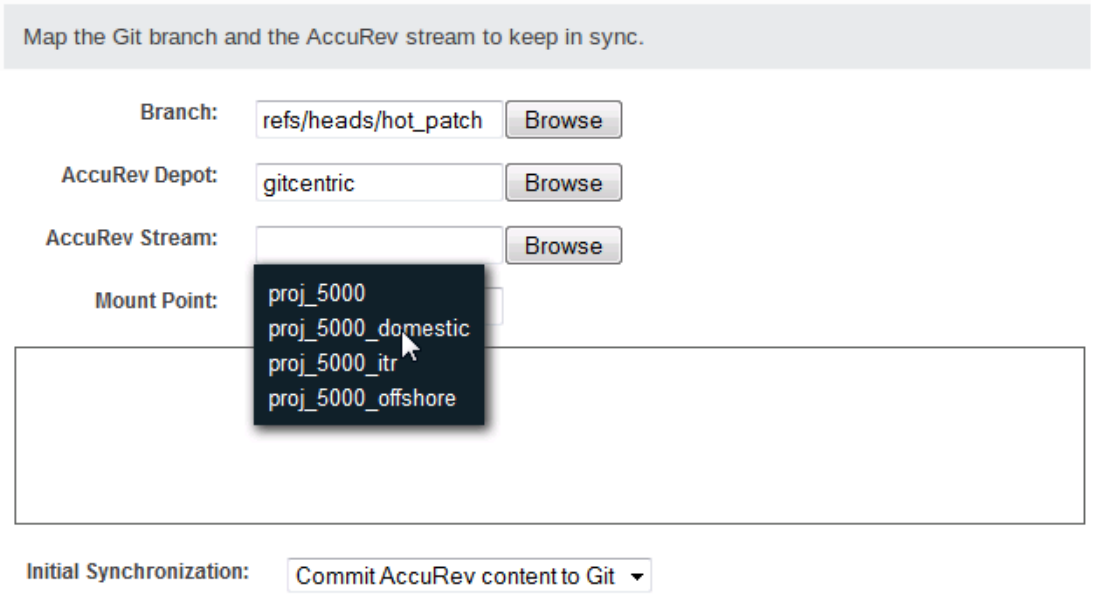
Once you have specified the AccuRev server connection, you can map Git branches to AccuRev streams.

Note: You cannot map to the root stream of a depot.

1. In the **Branch-Stream Mappings** section, click the **Add** button.



GitCentric opens a panel beneath the table, as shown in the following illustration.



2. In each of the fields in this panel, you can click a **Browse** button to navigate to the value you wish to specify. Most fields are self-explanatory, but the Mount Point warrants additional discussion.

   The *Mount Point* is the directory within the stream that will synchronize AccuRev and Git content. GitCentric provides a display that allows you to navigate to the desired directory. You can select the

root of the file system within this stream, or a subdirectory, which populates the field. The folder you select determines which AccuRev-controlled files will populate your repository.

Note: This step assumes that you have carefully planned out your branch-to-stream-mapping, and that you have either created a new stream or have identified an existing stream that is appropriate for mapping. See *Keeping Git Merges and AccuRev Promotes in Sync* on page 5 for more information.

3. Specify in which direction the files should flow when the Git repository and the AccuRev stream are first mapped. If you are mapping a repo with content to a new stream, select **Commit AccuRev content to Git**. If you are mapping an existing stream with files to a new repository, select **Commit Git content to AccuRev**.
4. Repeat these steps for each branch and stream that you want to map.
5. When you are done, click **Save** to save the branch-stream mapping. The next time you view the AccuRev stream, a "G" icon will be displayed for every stream that you have mapped to a GitCentric repo via the AccuRev Connector.

Test Your Mapping Status

When you save your mapping, GitCentric synchronizes the contents of the Git branch and the AccuRev stream by importing (branch-to-stream) and exporting (stream-to-branch) files as necessary. The status summary appears in the **Status** column of the Branch-Stream Mappings table. The status is updated every five seconds. When it displays *Idle*, the synchronization has completed.

To view details of this synchronization, including information about whether the import succeeded or failed, click the **View Details** button to display the Status Monitor. See *Status Monitor* on page 69 for more information. If the synchronization failed, see *Troubleshooting* on page 42 for pointers on addressing any errors.

Create a Clone and Test It

At this point, your Git users can create clones of the repo (within the limitations of whatever security settings you define). See *Create a Clone From a GitCentric Repository* on page 19. Whenever new content

is pushed to the repo, it will automatically be reflected in the mapped AccuRev stream. On the AccuRev side, any content that is promoted into the mapped stream will automatically appear in the repository.

Troubleshooting

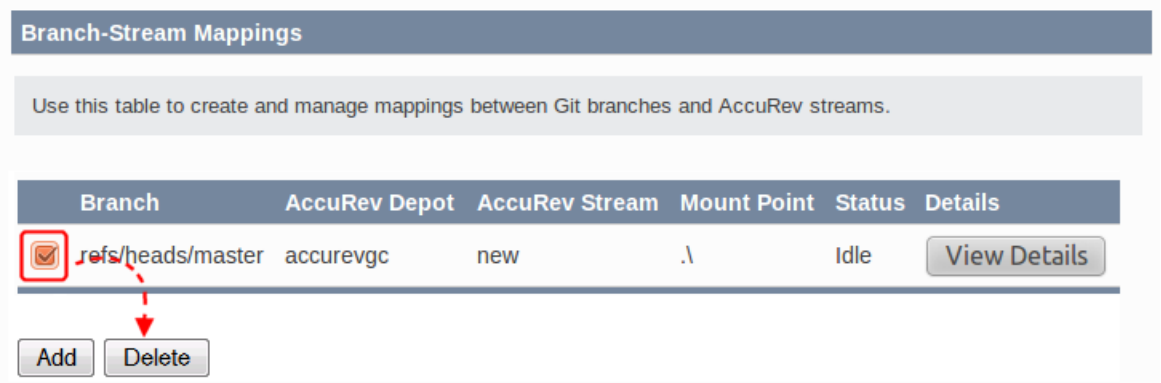
If the initial import/export operation failed after the mapping, check the following:

- Check **Administration -> Repositories -> *<repoName>*-> AccuRev Connector/Branches** for the status and the current branch SHA.
- Check the Status Monitor for import and export failure messages.
- Use the kandoMaintain command (see *Appendix A The kandoMaintain Utility*) to enable PRESERVE_TEMP_FILES and ACCUREV_COMMAND_LOG debug configuration settings, and retry, running AccuRev commands manually to debug.

Unmap a Git Branch from an AccuRev Stream

Unmapping a Git branch simply removes the association between the Git branch and the AccuRev stream, which prevents any future synchronization between the two. Unmapping does not remove any content from either the branch or the stream.

1. Log in to GitCentric as described above in *Log In to GitCentric* on page 14, specifying the server which has the stream to which the branch is mapped.
2. Click the **Administration** button, then click the **Repositories** menu to display the Repositories page. (See *Chapter 5 Administration* for reference information.)
3. Click the repository whose branch you want to unmap.
4. On the Repositories page, click **AccuRev Connector** menu. (Reference information is a available at *AccuRev Connector* on page 68.)
5. From the list of mapped branches, identify the one that you want to unmap, then click its checkbox and then click **Delete.**



Enable and Use Change Packages

Change packages is an AccuRev feature that associates changes made to an element with a specific issue in an issue tracking system; users are required to specify that issue number each time they promote changes out of a workspace. To enable change packages for your Git commits you need to do the following:

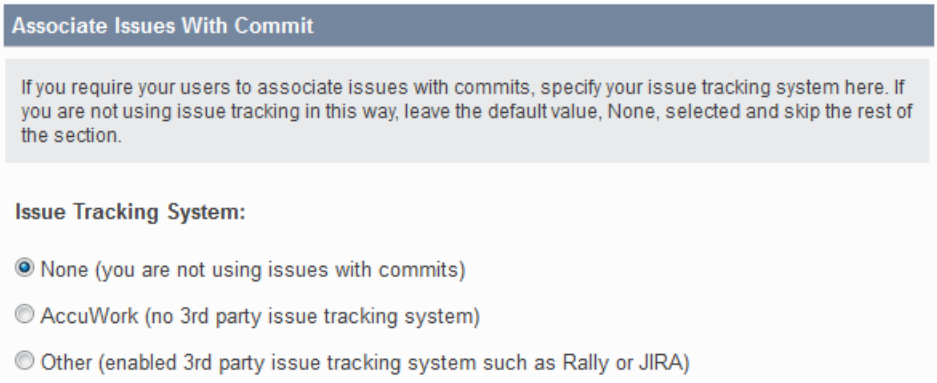
- In AccuRev, ensure that change packages are configured on the relevant AccuRev server. See the AccuRev on-line Help, which includes all the current AccuRev documentation. Use the index and search mechanisms in the on-line Help to learn how to configure change packages, and optionally, third-party issue tracking systems (ITS) if you are using something other than AccuWork.
- In GitCentric, specify the ITS you are using and the format of the Git commit message you will require users to follow when pushing commits. You specify this information on the **AccuRev Connector** panel of the Repositories page as described in this section.

Once you have enable change packages, you must inform your users of the requirement to add comments to their commits in the format you choose. The process for using a regular expression to specify the commit message format is described in the following section.

Specifying the Commit Message Format

To specify the commit message format that users will be required to follow when pushing commits (and from which the GitCentric bridge will obtain the issue number required to enforce AccuRev change packages):

1. Click the **Administration** button, then click the **Repositories** menu to display the Repositories page. (See *Chapter 5 Administration* for reference information.)
2. Click the repository for which you want to enable change packages.
3. On the Repositories page, click **AccuRev Connector** menu and locate the **Associate Issues With Commits** section. (Reference information is a available at *AccuRev Connector* on page 68.)



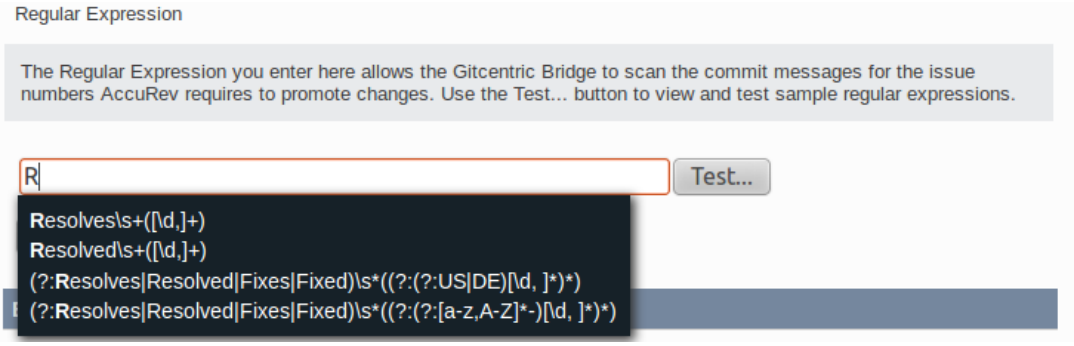
4. In the **Issue Tracking System** field, specify whether you are using AccuWork or a third-party issue tracking system.

   Tip: If this field is disabled, specify the AccuRev server connection information and click the **Save** button.

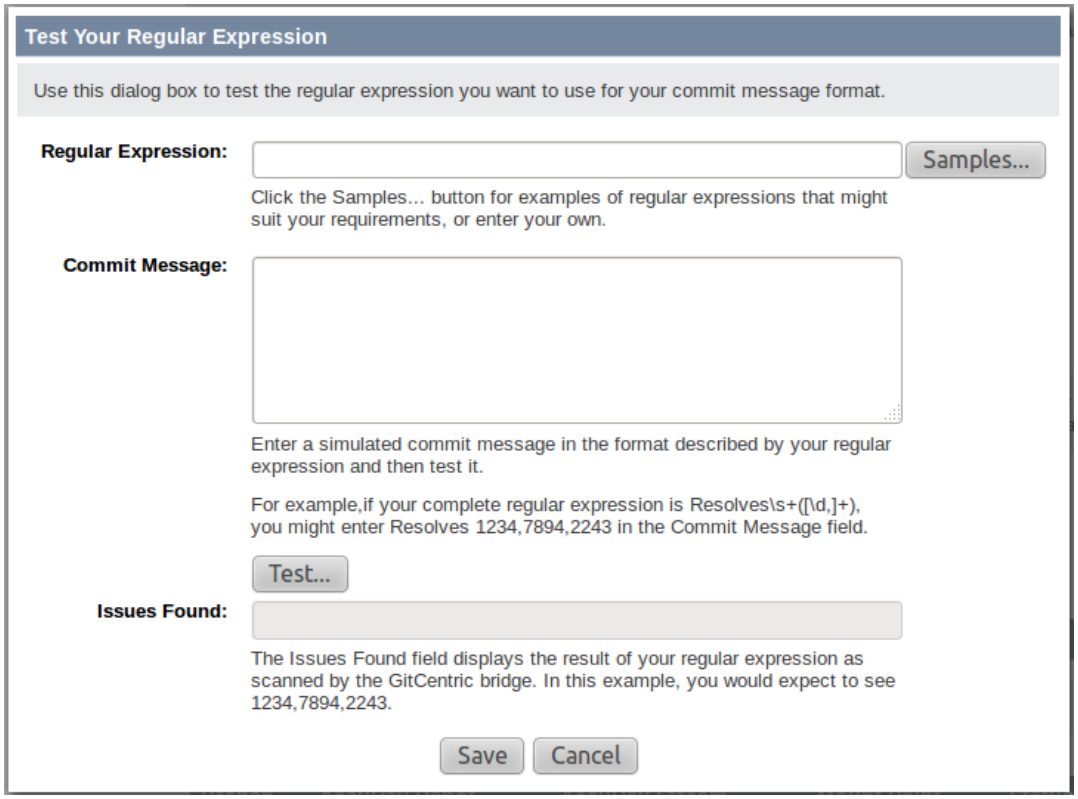
5. In the **Regular Expression** field, use a regular expression to specify the format of the commit message you require -- *Resolves issue 1234*, for example. You can accomplish this by either:

   - Entering a regular expression.

Tip: GitCentric displays sample expressions as soon as you begin typing in the field, as shown in the following illustration.



- Clicking the **Test...** button to display a dialog box that lets you experiment and verify that the regular expression you enter satisfies your requirements.



Refer to the instructions on the dialog box for its use.

Clicking the **Save** button on this dialog box saves your regular expression to the **Regular Expression** field in the **Associate Issues With Commit** section of the AccuRev Connector panel.

6. Click the **Save** button in the **Associate Issues With Commit** section to save your work.

   See *About GitCentric Regular Expressions* for more information on regular expressions and their use in GitCentric.

About GitCentric Regular Expressions

One of the powers of AccuRev is to associate all of the files affected by a particular change to an issue number tracking that change. This grouping of affected files is called a *change package*. (See the AccuRev documentation for information about change packages.) If you have change packages enabled on your AccuRev server, either through AccuWork or through a third-party issue tracking system, you may need to enforce the practice of including the issue number in the comment related to Git commits. That is the purpose of the **Regular Expression** field: to specify a commit message format that users must follow when pushing a change in Git that will be synchronized to AccuRev. The default example provided in the UI (`Resolves\s+([\d,]+):`) forces users to start their commit messages with "**Resolves *<issue number>*:**". If this is not found, the push will be rejected.

A discussion of Java regular expressions is beyond the scope of this document, but here are examples of what comments would be valid for some of these predefined expressions:

`(?:Resolves|Resolved|Fixes|Fixed)\s*((?:(?:US|DE)[\d, ]*)*)` -- *This format would be useful for the Rally issue tracking system, which makes use of user stories ("US") and defects ("DE"): The comments can start with "Resolves", "Resolved", "Fixes", or "Fixed", followed by white space, and then any number of issue numbers prefixed with "US" or "DE":*

Resolves US302, DE1405, US27
Fixed DE12849

`Resolved\s+([\d,]+)` -- *This format would be adequate for simple environments where the comment always starts with "Resolved", followed by white space, followed by any number of issues, separated by commas:*

Resolved 12576
Resolved 34, 149, 11057, 686

Use a search engine and search for "Java regular expressions" for more information about these formats.

Troubleshooting Change Package Errors

Your users may see change package error messages upon a push, and they must contact you as the GitCentric administrator for assistance in rectifying them. Errors are generated when GitCentric synchronizes the push with the AccuRev server, and the AccuRev server encounters a problem with the elements that comprise the change package. You (the GitCentric administrator) must act as a liaison to the AccuRev side of the world, including understanding change package errors and helping your end-users correct them. The following table summarizes some errors you might encounter and what you can do to address them.

Table 5. Addressing Change Package Errors

Error	Problem	Action
Change Package Required	An issue must be included in the commit comments, in the format dictated by the regular expression that was specified when change packages were enabled	Use `git commit --amend` to effectively rewrite the commit message.
Issue did not meet criteria	Typically means that the commit comment was present and generally formatted correctly, but the state of the AccuWork issue was incorrect (for example, "Completed" instead of "WIP")	Use `git commit --amend` to effectively rewrite the commit message.

Table 5. Addressing Change Package Errors

Error	Problem	Action
cpk merge required	Results during parallel development when two users make changes on the same file related to the same issue and one user tries to push his or her changes before pulling the other's, creating the merge requirement.	To address an existing error, you will need to create a new issue and push changes against it. To avoid such problems in the future, instead of using a simple **git pull**, use **git pull --rebase** to provide serialized commits, avoiding the need to separately merge remote changes in your branch.

Note that if the **push** command proceeds far enough to scan for issues, it outputs the issue it has detected in the commit message ("Issue(s):" ... for example). If the **push** command fails, reviewing this output may provide information that can help you troubleshoot any problems with the regular expression that was used to specify the commit message.

For understanding and correcting change package errors that are caused beyond the GitCentric regular expression filter, please consult the AccuRev on-line help, which includes all AccuRev documentation. Use the search mechanism for change package discussions, particularly in the *Concepts Guide* and *Technical Notes*.

Add an AccuRev Server

If the AccuRev server you use for SCM is the same one that the original Administrator user logged into after installation, you may never need to perform the procedures described in this section. But if you have multiple AccuRev SCM servers, then you need to add them to GitCentric.

> **Note:** A client pointing to each master AccuRev server must be on the same machine as the GitCentric server.

Adding an AccuRev server is a two-step process:

1. **Register the AccuRev server with GitCentric.** You need to register an AccuRev server with GitCentric if:
   - You are adding another AccuRev server to GitCentric
   - Your existing AccuRev server has multiple IP addresses

   See *Registering an AccuRev Server* on page 46.
2. **Configure the server** as described in *Configuring the AccuRev Server* on page 47.

Registering an AccuRev Server

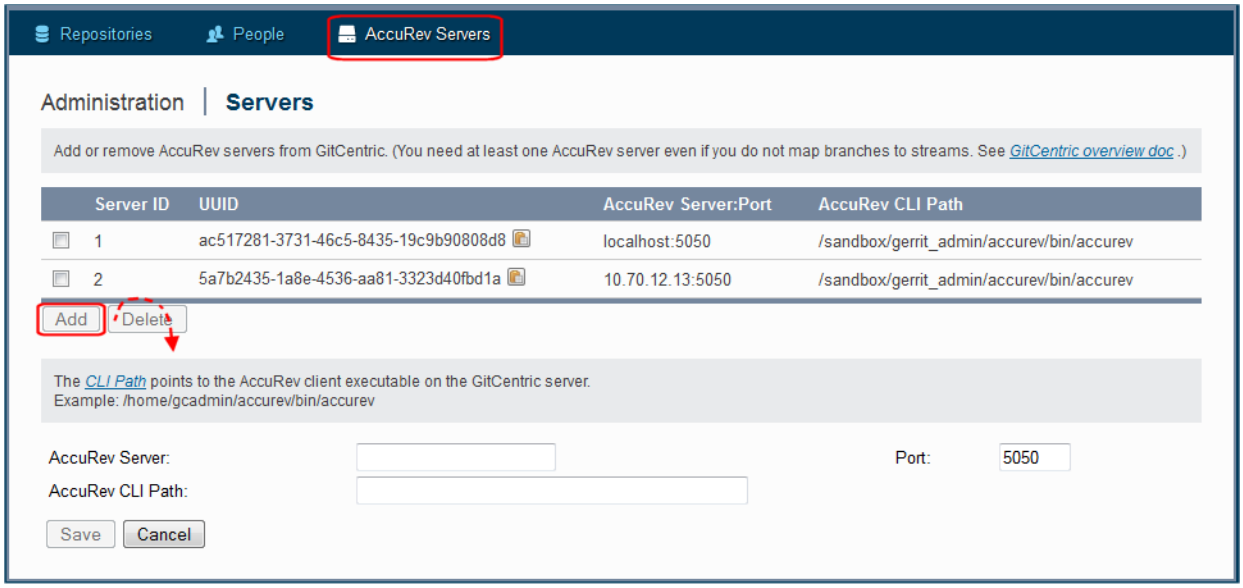
For security reasons, the GitCentric bridge accepts commands only from IP addresses that it recognizes. If you add an AccuRev server that has multiple IP addresses (for example, if the server has multiple Ethernet cards), you must register each IP address. In this case, you should perform GitCentric operations only against one of these servers entries -- typically one where you have specified a host name rather than an IP address.

To register an AccuRev server:

1. Log in to GitCentric as described in *Log In to GitCentric* on page 14.

2. Click the **Administration** button, then click the **AccuRev Servers** menu to display the Servers page. (See *Chapter 5 Administration* for reference information.)
3. Click the **Add** button.

   A panel appears to allow you to register the AccuRev server.



4. The **AccuRev Server** and **Port** fields are relatively self-explanatory, but the **AccuRev CLI Path** may need careful reading:

   - **AccuRev Server:** -- The host name or IP address of the AccuRev server you wish to associate with one or more repos, or the IP address for an AccuRev server with multiple IP addresses. (The name of the server you first logged in to GitCentric with -- see *Log In to GitCentric* on page 14 -- is automatically added to the GitCentric database.)

     Note that if you specify `localhost`, this indicates an AccuRev server on the GitCentric host, not necessarily the machine where you are running your browser.
   - **Port:** -- This is typically "`5050`" for most AccuRev servers, but the AccuRev administrator may have changed it to something different.
   - **AccuRev CLI Path:** -- This is the path to the AccuRev client executable on the GitCentric server that should be used to communicate with a particular AccuRev server. For example:

     `<ac_home>``/bin/accurev`

     For more information, see *A Note about the CLI Path Setting* on page 74.
5. Click the **Save** button.

   GitCentric refreshes the **Servers** table with the server you just added.

Configuring the AccuRev Server

The previous procedure simply registers the AccuRev server with GitCentric. Before you can use this server with GitCentric and Git, you must configure the server using the following procedure.

1. Check to see if a `triggers` directory exists under *`<ac_home>`*`/storage/site_slice/`. If not, create it.

2. Copy the `server_master_trigger.pl` file from *<gc_home>*`/bin` to the AccuRev server trigger directory. For example:

   ```
   > cd <gc_home>/bin
   > cp server_master_trig.pl <ac_home>/storage/site_slice/triggers
   > cd <ac_home>/storage/site_slice/triggers
   > chmod +x server_master_trig.pl
   ```

   Note: `server_master_trig.pl` includes several lines of documentation in comments toward the end of the file.

3. Locate the `################ ACCUREV CLIENT PATH` section in the `server_master_trig.pl` file and specify the path for your AccuRev client. For example:

   ```
   $::AccuRev = "/usr/accurev/bin/accurev";
   ```

4. Log in to AccuRev.

   Tip: Log in using the **-n** option so that the user's log in session does not expire.

5. See the *GitCentric Installation and Release Notes* for suggestions about testing newly-configured servers.

Configure Multiple AccuRev Servers

You can configure additional AccuRev servers for GitCentric using the same procedures above. However, you should observe the following guidelines:

- Use the same username and password across all servers. Otherwise, you will frequently need to re-enter your login credentials.

  If the multiple AccuRev servers are running different AccuRev versions, you must have an AccuRev CLIENT for each version installed on the GitCentric server. When you configure a server, you must point to the correct client in the "CLI Path" field. For more information, see *A Note about the CLI Path Setting* on page 74.

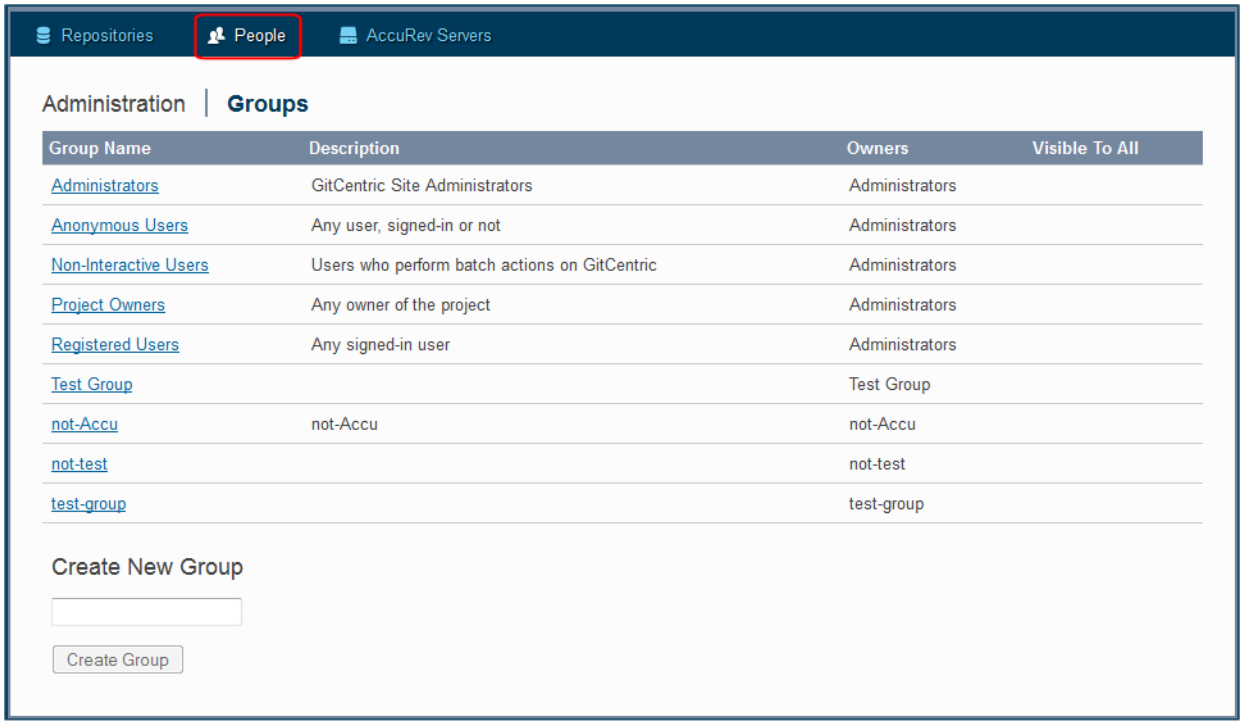
- If you have a situation where IP addresses can change over time (such as dynamic IP addresses assigned via DNS), you may need to consider setting `AC_BRIDGEAPI_SECURITY_POLICY` to `AllowAnyHost`. Note, however, that this workaround has a security impact that you need to carefully consider before implementing it. See *GitCentric Bridge Configuration Settings* on page 79 for more information.

Manage GitCentric Groups

GitCentric groups are derived from Gerrit Code Review groups. See the *Gerrit Code Review documentation* for details about how groups are implemented. The following section describes how to administer groups in the GitCentric context.

To View and Access Groups

Groups can be viewed, defined, and modified by clicking **People** from the GitCentric **Administration** menu:



Add a Group

To add a group to GitCentric:

1. Click the **Administration** button, then click the **People** menu to display the Groups page. (See *Chapter 5 Administration* for reference information.)
2. Enter a name in the **Create New Group** field on the Groups panel.
3. Click **Create Group**.

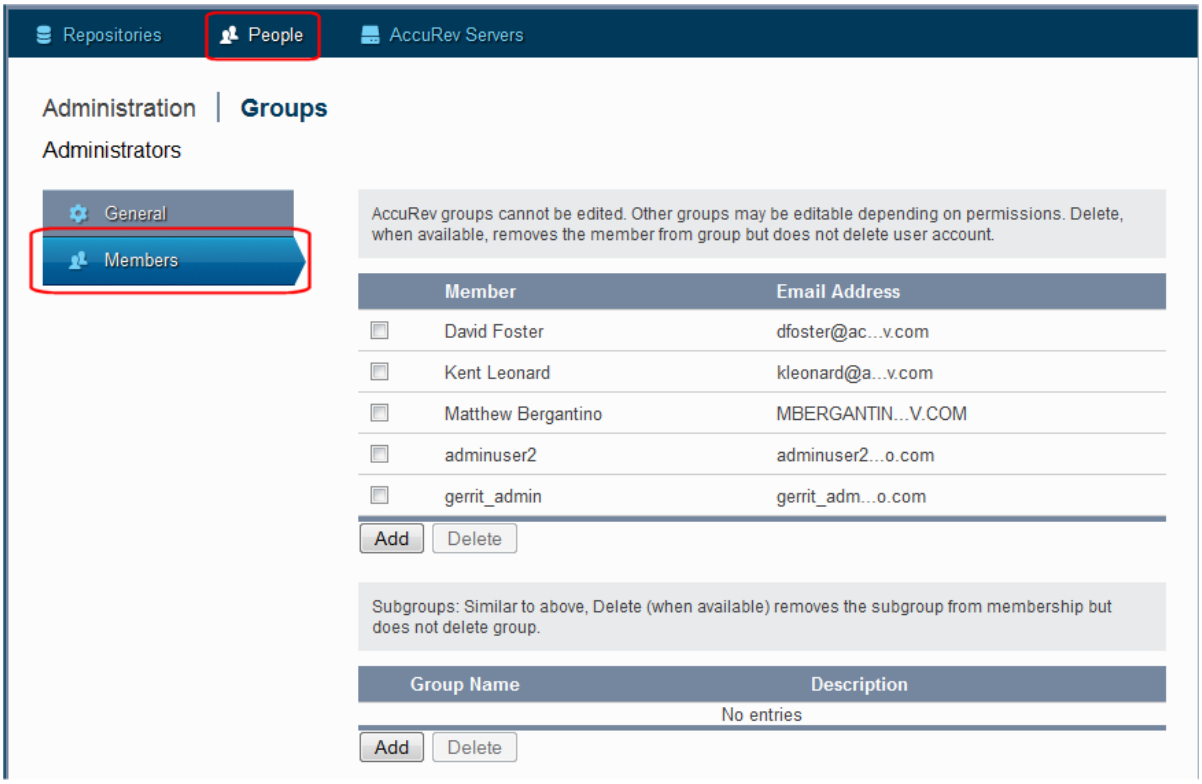
Add a Member to a Group

Note: Any member you wish to add to a group must exist in the AccuRev Server with which GitCentric was configured during installation. You cannot use this procedure to *create* members.

To add a user to a GitCentric group:

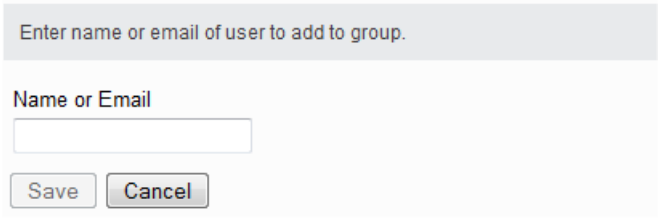
1. Click the **Administration** button, then click the **People** menu to display the Groups page. (See *Chapter 5 Administration* for reference information.)
2. Click the group to which you wish to add a member from the Group Name column.

   The page is refreshed to show general content.
3. Click the **Members** menu.



4. Click the **Add** button for the members table.

   A new panel appears to allow you to specify the user by name or email:



5. Start to enter a value in the **Name or Email** field. GitCentric provides type-ahead choice of valid users.
6. Select the user you wish to add and click **Save**.

AccuRev Groups

AccuRev groups are a special case handled by GitCentric.

If your account is a member of a group in AccuRev, that group will be created in GitCentric as soon as you log in, and the group will be named with the format: `Accurev.<accurevGroupName>`

Note that AccuRev group hierarchies are not replicated in Git Centric: if you belong to “Group1” and also to a subgroup of Group1 named “Group2:, GitCentric creates two peer-level groups: `Accurev.Group1` and `AccuRev.Group2`.

If you have upgraded to GitCentric from the Kando 2012.1 release, you probably have a `kando_admin` administrators group on your AccuRev server. If so, when GitCentric creates the Accurev.kando_admin group, it automatically grants this group "Administer Server" permission. You can check this using the Global-Capabilities section on the Repositories page. (Click the **Administration** button, then click **Repositories**. On the Repositories, page, click the **All-Projects** repository and the **Access** menu.)

> **Note:** If you have multiple AccuRev servers, groups with the same name from all AccuRev servers are combined in GitCentric.

AccuRev groups created in GitCentric are read-only in the UI: you can see the properties and current members, but you cannot modify them.

Membership in an AccuRev group in GitCentric is only updated upon login. If you change a user's group membership in AccuRev, it will not change for the GitCentric user until that user logs out (either explicitly, or due to a session time-out) from Git Centric and then logs back in.

Configure AccuRev Element ACLs (EACLs)

You can configure Element ACLs (EACLs) on the AccuRev server for the GitCentric Service Account, to determine which files can be edited in the GitCentric repository. Files that cannot be edited do not appear at all in the GitCentric repository.

To set up AccuRev Element ACLs, see the following AccuRev documentation:

- The “Security” chapter in the AccuRev *On-Line Help*
- The **eacl** command description in the AccuRev *CLI User’s Guide*
- The “Element-Level Security (EACLs)” section in the "AccuRev Security Overview" chapter of the AccuRev *Administrator’s Guide*

Enable/Disable Code Review

To enable GitCentric's optional Gerrit Code Review functionality:

1. The ACLs required to support code review are configured automatically for all new GitCentric installations; users *upgrading* to GitCentric 2013.3 may need to configure additional ACLs. As there is no one set of ACLs that is appropriate for every installation, you, as the administrator, should review these settings and adjust them as needed. See *Configuring ACLs for Code Review* on page 36 for more information.
2. Your users must configure their clones by adding the following line to the `[remote "origin"]` section of each clone's `config` file:

   ```
   push = refs/heads/*:refs/for/*
   ```

   This specifies the source (`refs/heads/*`) and target (`refs/for/*`) branch mapping that will be used to push changes to a special branch for code review. Users can add this line by executing the following command on each clone:

   ```
   git config remote.origin.push 'refs/heads/*:refs/for/*'
   ```
3. Copy the `commit-msg` hook to the `.git/hooks/` directory of each clone. For example:

   ```
   scp -p -P <port><server>:hooks/commit-msg .git/hooks/
   ```

   where:

   `<port>` is the SSHD listener port (typically `29418`)

   `<server>` the host where GitCentric is installed

If you forget to copy the `commit-msg` hook, your push will fail. The error message you receive in this situation instructs you to copy the `commit-msg` hook, as described in *Step 3* above. You must then execute a

```
git commit --amend
```

before trying the push again.

Allowing Self-Reviews

By default, Gerrit Code Review does not allow self-reviews. If you want to configure Gerrit Code Review to allow self-reviews, you must do so in the Gerrit Code Review Prolog file, rules.pl. In new installations of GitCentric 2013.3, a copy of this file is installed to the `/meta/config` branch of the special "All-Projects" repository.

Working with the rules.pl Prolog file is beyond the scope of this document. Refer to the Gerrit Code Review documentation for more information:

https://gerrit-review.googlesource.com/Documentation/prolog-cookbook.html

Disabling Code Review

Disabling code review is a two-step process:

1. As the administrator, clear the **Enable Code Review** check box on the General panel for the repositories for which you wish to disable code review.

   For *new installations* of GitCentric 2013.3 and later, this action automatically configures the repository's ACLs to enable direct push. If you have *upgraded* to GitCentric 2013.3 and later, this field appears as `Advanced...` and you must configure the repository ACLs to enable direct push manually. See *Configuring GitCentric ACLs for Direct Push* on page 38.

2. Individual users must remove or comment out the following line in the `[remote "origin"]` section of each clone's `config` file:

   ```
   push = refs/heads/*:refs/for/*
   ```

Enable Gerrit Code Review Replication

Gerrit Code Review can be configured to push changes from Git repositories to one or more remote hosts. This section describes how to enable Gerrit Code Review replication in GitCentric. For general concepts related to Gerrit Code Review replication, refer to the Gerrit Code Review documentation.

Overview

The basic procedure for enabling Gerrit Code Review replication is summarized here:

1. Register the SSH public key representing the GitCentric bridge account with the Gerrit Code Review administrator's account.
2. Ensure that the server running Gerrit Code Review has SSH access to the server to which changes will be replicated. If the machine running Gerrit Code Review has never connected to the replication server, it will be listed as an unknown host and Gerrit Code Review will close the connection.
3. Modify the sample Git-style configuration file, replication.config, that is installed with GitCentric.
4. Stop and restart Gerrit Code Review.

The following sections provide additional detail.

Registering the GitCentric Bridge SSH Key with Gerrit

The SSH public key representing the GitCentric bridge account is registered automatically when you install GitCentric the first time. If you have upgraded an existing GitCentric installation, you need to register the SSH public key manually. See *Public Keys* on page 59 for more information.

Modifying the replication.config File

Gerrit Code Review uses a replication.config file to identify the URL of the server to which changes will be replicated. For example:

```
[remote "host-one"]
  url = gerrit2@host-one.example.com:/some/path/${name}.git
```

A sample replication.config is installed to <gc_home>site/etc. To use this file:

1. Uncomment the example.

   Tip: Consider copying the example and using that copy to specify the URLs for one or more remote hosts.

2. Change the sample values for values applicable to your environment.
3. Stop Gerrit Code Review, which you can do by stopping the Tomcat web server. For example:

   ```
   cd <ac_home>/WebUI/tomcat/bin
   ./shutdown.sh
   ```

4. Restart Gerrit Code Review. For example:

   ```
   cd <ac_home>/WebUI/tomcat/bin
   ./startup.sh
   ```

   Note: Make sure that the user starting Tomcat has write access to the logs, temp, webapps, and work directories in <ac_home>/WebUI/tomcat. This user should have read access to all other Tomcat directories and files.

Troubleshoot Import/Export Operations

If you encounter an issue with syncing between a Git repository and AccuRev, check the following.

AccuRev to Git

- Check the triggers.log and ensure that the trigger has fired.
- Check the KandoBridge.log and ensure that the trigger is reaching the bridge.
- Test server_master_trig.pl as described in the *GitCentric Installation and Release Notes* to check for missing dependencies.
- Check that all necessary processes are running: postgres, tomcat, accurev_server.
- Retry by "tickling" the stream: `accurev chstream -E t -s <stream>`.

Git to AccuRev

- Check for any `git push` error messages.
- If you receive the following error, particularly when first mapping an existing Git repository to an AccuRev stream, you may be trying to map a case-sensitive Git repo to a case-insensitive AccuRev depot:

  ```
  Import: com.accurev.gemini.commonapi.AHCommonAPIException Element already exists
  ```

  See the Note: under *Import an Existing Git Repo* on page 31 for more information.
- If you are using Gerrit Code Review, make sure that the change is submitted (marked as review passed) after `push`.
- If you are using Gerrit Code Review and the change state is 'merge failed' then the import failed. Import errors (which are usually `push` errors) appear in the Comments table at the bottom of the Gerrit Code Review Change page.
- Check the kandoGerrit.log and ensure that the bridge is being called.
- Check the kandoBridge.log and ensure that hooks are reaching the bridge.
- Check that all processes are running: postgres, tomcat, accurev_server.
- Check the AccuRev `acserver.cnf` configuration file and ensure that the service account group (typically "`scm_bridge_group`") is set as the value for `ASSIGN_USER_PRIVILEGE,` and that the group includes the actual user account (typically "gcSyncUser") as a member, and that this user account is the one that was specified when the branch was mapped to the AccuRev stream. (**Note**: some installations may use `CC_USER` rather than `ASSIGN_USER_PRIVILEGE`.)
- Retry by redoing the `push`.

Set Up Gerrit Garbage Collection

Gerrit Code Review garbage collection (`gerrit gc`) is a command that converts loose objects to packed, and that also removes unused objects. Micro Focus recommends that you run garbage collection on a regular basis to avoid performance problems associated with loose objects that occur over time.

This topic provides general examples of cron jobs you can use to set up garbage collection for your site. Note that syntax and supported features for cron jobs can vary across platforms. For more information, refer to cron and crontab documentation for your platform.

Tip: Consider running garbage collection on all of your projects on a regular basis. Choose a "quiet" time that will be unlikely to interfere with repository users.

Gerrit gc Syntax

The basic syntax for the Gerrit garbage collection command is:

```
ssh -p <port> <user>@<host> gerrit gc [--all] [ <project> ... ]
```

where:

<port> is the GitCentric port number, typically `29418`.

<user> is a user with Gerrit "Administrate Server" privileges.

<host> is the name of the GitCentric host machine.

`--all` runs garbage collection on all projects in the repository.

<project> is the name of one or more individual projects in which you want to run garbage collection.

cron job Examples

You use cron jobs to execute commands, like `gerrit gc`, on a scheduled basis. The cron job format is a sequence of five fields used to specify day, time, and frequency; the sixth is used to specify the command:

Table 6. Cron Job Format

Field	Description	Allowed Values
MIN	Minute of the hour	0 to 59, *
HOUR	Hour of the day	0 to 23, *
DOM	Day of the month	1 to 31, *
MON	Month	1 to 12, *
DOW	Day of the week	0 to 6; *; sun, mon, tue, wed, thu, fri, sat
CMD	Command	The command to be executed

The following examples show how you can use cron jobs to run garbage collection on your Gerrit projects. In these examples, "comet" is the name of the GitCentric host machine.

Run garbage collection on all projects at midnight every day of the week.

```
0 0 * * * ssh -p 29418 admin@comet gerrit gc --all
```

Tip: You can use the keyword `@daily` to express `0 0 * * *`.

Run garbage collection on all projects weekly at midnight Sunday.

```
0 0 * * sun -p 29418 admin@comet gerrit gc --all
```

Tip: You can use the keyword `@weekly` to express `0 0 * * 0` (which is the equivalent of `0 0 * * sun`).

Run garbage collection on "acme" and "phoenix" projects Monday through Friday at 1:30 a.m.

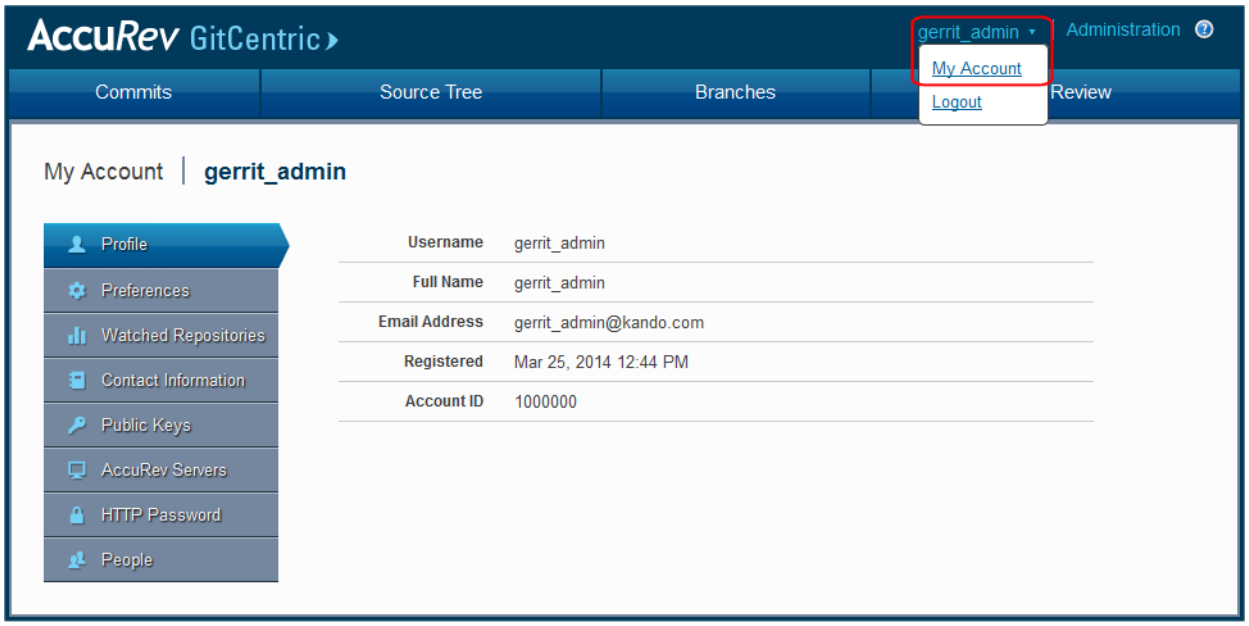
```
0 0 * * 1-5 ssh -p 29418 admin@comet gerrit gc acme phoenix
```

3. My Account

This chapter describes the features of the GitCentric **My Account** page.

Opening the My Account Page

To open the **My Account** page, choose **My Account** from the username drop-down menu:



Menu Options

The following sections summarize the features of the **My Accounts** menu.

Profile

Displays a non-editable summary of your user information. You can change the **Full Name** and **Email Address** values through the **Contact Information** panel described below. The **Username** value shows your AccuRev account name. The **Account ID** is assigned to you automatically when you register with GitCentric. Note that ID 1000000 is always the first user to register with GitCentric, and is automatically added to the Administrator users group. Note that while the information of the **Registered** field cannot be modified, its display format can be controlled with the **Date/Time Format** option under **Preferences** (see below).

Preferences

The Preferences page provides access to display and usability options.

Field:	Description
Show Site Header	Enables/disables the optional header and footer images in your web UI display. See *Define and Display a Site Header or Footer for Gerrit Code Review* on page 18.
Use Flash Clipboard Widget	When enabled, displays a widget on various text fields throughout the web UI that allows you to copy the contents of the field to the clipboard:  **Note:** Requires a Flash-enabled browser.
CC Me on Comments I Write	When enabled, you will receive an email containing comments that you submit in response to a review request. The default is to have this setting disabled, to reduce the amount of code review email you receive. But if you want e-mailed confirmation of comments published by your account, you have the option of enabling this setting.
Display Patch Sets in Reverse Order	When enabled, reverses the display order of the patch sets in the Change Screen so that the latest patch set is always shown on top. This is useful when there are several patch sets for a change, and the latest patch set and the links to the diffs in the patch set end up below the fold of the Change Screen.
Display Person Name in Review Category	When enabled, displays the name of the last reviewer in the "R" column of the Gerrit Code Review Dashboard.
Maximum Page Size	Specifies the maximum number of rows that will be visible in Gerrit Code Review displays.
DateTime Format	Allows you to specify one of four Date formats (three American, with month then day, using commas, hyphens, or slashes; or one European, with day then month, separated by periods), and one of two Time formats (12 hour AM/PM, or 24 hour).

Watched Repositories (Projects)

> **Note:** Gerrit Code Review often uses the term "Project" to refer to a repository. AccuRev uses the terms "repository" or "repo" when referring to a repository, and reserves the term "project" when referring to specific Gerrit functionality, or when referring to some kind of planned or defined undertaking.

The Watched Repositories page allows you to identify the repositories whose changes you want to track. Click the **Add** button to add a repository to the Watched Repositories page and to optionally configure Gerrit Code Review to send an email notification whenever a change occurs in that repo. When adding a repo, use the **Browse** button to navigate to the repo of interest. If you want to specify conditions under which you want to be notified of a change (perhaps you are interested only in changes to a specific branch, for example), see the *Searching Changes* topic in the Gerrit Code Review documentation to learn more about search operations and search expression syntax.

Contact Information

Enter the name of the currently logged-in user. You can register multiple email addresses for this user, but only one can be specified as the “preferred” address. Gerrit uses the preferred email address when it needs to generate an email address for you; it uses others to validate incoming email to you.

Tip: GitCentric accepts mixed-case values in the email address domain. For example, both `johndoe@AcmeCo.com` and `johndoe@acmeco.com` are valid email addresses.

Public Keys

You must configure SSH public key authentication before you can upload changes.

If you already have an SSH key:

Click **Add** and paste the contents of `~/.ssh/id_rsa.pub` into the resulting **Paste the public SSH Public Key below:** field. Make sure that you do not introduce line-breaks when copy & pasting.

If you need to create an SSH key:

1. If necessary, install ssh-keygen (this is normally included as part of a Git or O.S. installation).
2. > `ssh-keygen -t rsa -C "<yourEmailAddress>@<yourDomain.com>"`

For more detailed information about SSH keys, see *Create an SSH key* on page 14, or follow the link on the dialog box.

AccuRev Servers

You use the AccuRev Servers page to associate a GitCentric user account with an AccuRev user account. You need to use this page only if you have multiple AccuRev servers in your environment. Otherwise, the GitCentric<-->AccuRev user account association is managed automatically, as part of the self-registration process. (See *Register with GitCentric* on page 16 for more information on self-registration.)

> **Note:** Although the UI does not enforce this, you should not specify multiple user accounts on the same AccuRev server here. If you specify more than one user ID on the same server, GitCentric may default to one that you did not intend. It is fine to specify multiple servers here, with one user ID each.

To specify a GitCentric<-->AccuRev user account association:

1. Click the **Add** button.
2. On the panel that appears:
   a. Select the server from the **AccuRev Server** drop-down list.
   b. Enter your username on that server in the **AccuRev Username** field.

      The AccuRev user on this server that you use for GitCentric functions. If you are an administrator, this might be an account like `acserver` if you have legacy AccuRev systems. See the *GitCentric Installation and Release Notes* for a discussion about this user account.
   c. Enter the password associated with that username in the **Password** field.
4. Click the **Save** button to register the account.

HTTP Password

Some environments do not permit you to use SSH to connect across a firewall. If you cannot use SSH, but you can connect to the server with smart HTTP, you can generate a password here, and then access your Git repositories with a URL constructed with "...p/*<repository>*".

Note: GitCentric supports only Git "smart HTTP" not so-called "dumb HTTP". Smart HTTP is documented on several Git-related sites.

The syntax to clone a repository with smart HTTP is:

```
git clone http://<username>:<password>@<host>:<port>/p/<repository>
```

Example:

```
git clone http://testuser:PaoYhDp8BFoA@localhost:8100/p/TestProj2
```

Once you have cloned, you can pull / push over HTTP just like you can over SSH.

Avoiding Password Entry

Generated passwords can be difficult to memorize, and providing one each time you connect using HTTP can be a nuisance. Fortunately, Git provides a number of mechanisms to help deal with this.

First, you can create a `.netrc` file in your home directory ("`_netrc`" if you are using msysgit on Windows). For example:

```
machine localhost
login testuser
password PaoYhDp8BfoA
```

The `.netrc` file lets you keep the *<username>*:*<password>* out of the URL for enhanced security, but it is still stored as plain text.

A more secure approach, if you are using Git 1.7.9 and later, is to use Git's *credential helper*, which tells Git to store your password in cache for 15 minutes:

```
git config --global credential.helper cache
```

You can increase the timeout by specifying the desired limit in seconds. This example shows the timeout set to 30 minutes:

```
git config --global credential.helper "cache --timeout=1800"
```

Use `--unset` to revert to specifying the password manually:

```
git config --unset credential.helper
```

People

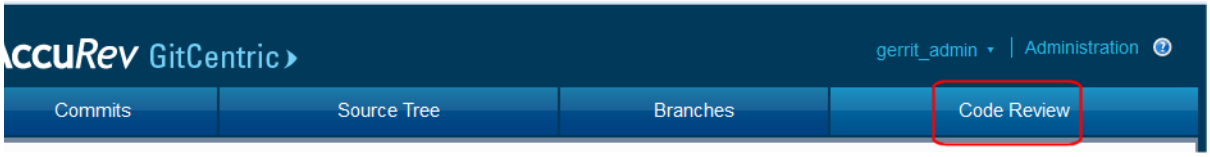
This is a read-only display of the security groups to which the logged-in user belongs. To create or modify groups, go to **Administration -> Groups** in the Web GUI. For details about how groups are implemented, see the *Gerrit Code Review documentation*. For "How to" information about managing groups in the GitCentric context, see *Manage GitCentric Groups* on page 49.

4. Code Review

This chapter describes the purpose of the GitCentric **Code Review** menu, which serves as a gateway to the third-party Gerrit Code Review tool. To learn how to use Gerrit Code Review in GitCentric, see *Enable/ Disable Code Review* on page 52.

Opening the Code Review Page

To open the **Code Review** page, click the **Code Review** tab:



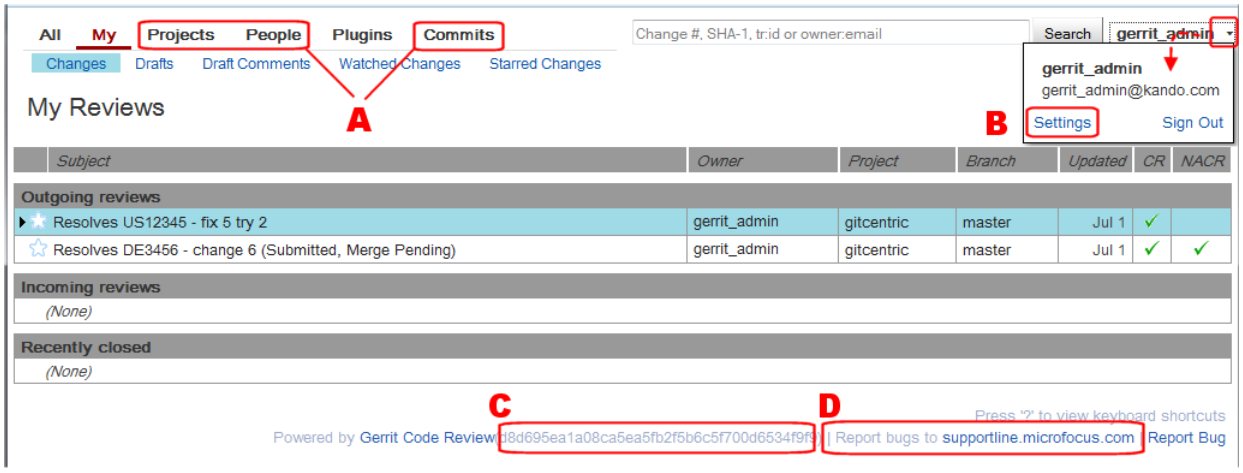
Overview of Gerrit Code Review

GitCentric incorporates the open source Gerrit Code Review tool, which provides web-based code review and project management for Git-based projects. GitCentric administrators can decide whether or not to require users to include Code review in their workflow. Even if your site does not require you to perform code reviews, GitCentric makes use of Gerrit's group-based ACLs and administration features.

Differences from Standalone Gerrit Code Review

You access Gerrit Code Review from the **Code Review** tab in the GitCentric UI (shown in the illustration at the top of the page). Log-in and registration are handled through the GitCentric UI (see *Log In to GitCentric* on page 14).

If you are familiar with Gerrit Code Review, you may notice the following differences from a standalone Gerrit Code Review installation.



Item	Description
A	Top-level menu items take you to the GitCentric GUI: • **Project** takes you to the Repositories page • **People** takes you to the Groups page • **Commits** takes you to the Commits page
B	The `Settings` link takes you to the My Account page in the GitCentric UI (see *Chapter 3 My Account*).
C	The SHA of the GitCentric build you are using.
D	Use this link to contact Micro Focus customer support about a GitCentric issue. Use the `Report Bug` link to contact Gerrit about issues with Gerrit Code Review.

Otherwise, all other Gerrit Code Review functionality is handled through the Gerrit UI, and documentation for Gerrit features is handled by the Gerrit on-line help. Gerrit Code Review has its own documentation, which you can find here:

http://gerrit-documentation.googlecode.com/svn/Documentation/2.7/index.html

Tip: Administration features are accessed from the GitCentric, and not the Gerrit, UI. (In GitCentric, the **Administration** menu takes you to the Administration page in the GitCentric UI; see *Chapter 5 Administration*.)

Code Review for Users of Differing Backgrounds

Experienced Git users: If you are a Git user who is new to both Gerrit and AccuRev, you will continue to use Git the same way you always have. However, if code review is enforced at your site, you may now need to configure your clones to push to a special code review branch (see *Create a Clone From a GitCentric Repository* on page 19). Your changes will need to be approved by others, and you may be requested to approve other people's changes. In any case, once your changes are approved and pushed to the repository, GitCentric automatically synchronizes them with the appropriate stream in AccuRev. Likewise, when you do a pull, fetch, or clone, you may find that changes have been synchronized from the AccuRev stream.

Experienced AccuRev users: If you are an experienced AccuRev user, your site may have you continue to manage your changes in AccuRev SCM. If so, you may never need to log into GitCentric or use code review. But if you start working in Git, you will use GitCentric to access the code review features, and to manage your account.

Experienced Gerrit users: If you are an experienced Gerrit user, not much has changed: You push your updates in Git, and they get code-reviewed in Gerrit. If you are a reviewer, you will log in through GitCentric, and then go directly to Gerrit where you will perform Gerrit operations the same way you always have. However, your changes will now be automatically synced between the repository and the AccuRev stream to which it is mapped. (And if changes are made in the AccuRev stream, they will automatically be synced with the corresponding repository, and you will see them the next time you do a **pull**/**fetch**/**clone** command.) The Settings link now goes to the GitCentric UI, and administrators will find that all administrative functions now appear through the GitCentric UI.

Regardless of your experience and your expertise, you should be aware that the four components that you may work with (Git, Gerrit, GitCentric, and AccuRev) each have their own documentation and knowledge base. Several excellent sites exist on the web for Git and Gerrit; and Gerrit, GitCentric, and AccuRev all

have extensive on-line help. And you should have at least one experienced user or administrator at your site who is knowledgeable about each of these technologies.

Gerrit Code Review and AccuRev Mappings

The mapping between Git repositories and AccuRev streams is generally transparent to Gerrit Code Review users, but you should be aware of the added meaning of the “Merged” status when a change has been approved: in a mapped GitCentric environment, merged not only indicates that the changes have been merged in the repository, but they have also been successfully imported to the AccuRev stream. If you cannot get to the “Merged” status, you should check for error messages on the Gerrit Code Review page and troubleshoot any issues preventing the import.

Troubleshooting

If you have difficulty submitting an approved change and cannot import it to AccuRev (as indicated by a “Merge Pending” or some other status), check the following:

- Look for error messages in the “Change comments” area
- Have an administrator check that the following group ACLs are correct for the repository: Submit, Label Code Review, and Label Verified
- Ensure that the commit parentage is up to date, and rebase if necessary
- Examine the dependencies list for any changes that need to be submitted before the current change will merge
- Check the kandoGerrit.log for errors
- Check the kandoBridge.log for errors

5. Administration

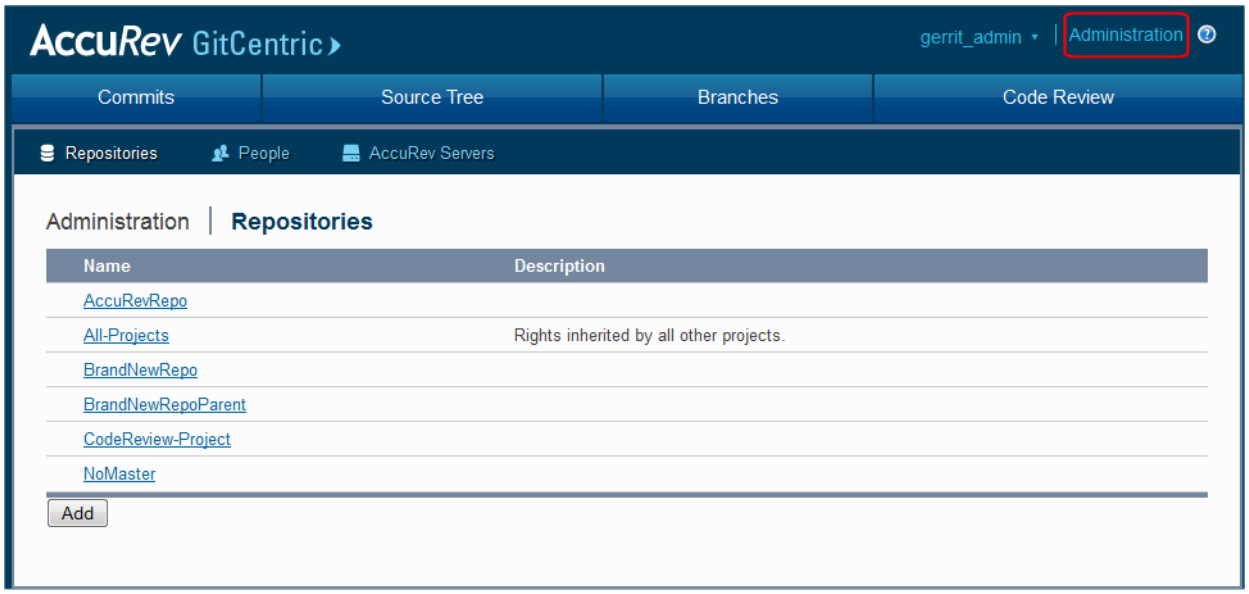
This chapter provides reference information for the features of the **Administration** page. The Administration menu allows you to configure:

- repositories
- groups
- AccuRev servers

For task-oriented procedural information about using this page, see *Chapter 2 How to...*.

Opening the Administration Page

To open the **Administration** page, click the **Administration** button in the upper right corner of the GitCentric UI:



Repositories

The Repositories page (shown in the preceding screenshot) lists the repositories that are currently registered with GitCentric, and provides the ability to create a new one.

To Create a New Repository

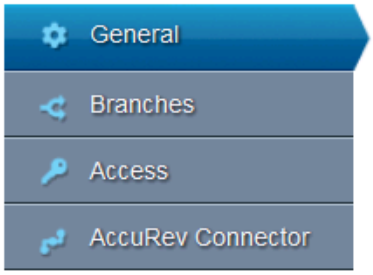
Click the **Add** button to create a new repository. Procedures for using this panel are found at *Create a Repository for GitCentric* on page 29.

Table 1. Create repository options

Field	Description
Repository Name	Enter any valid string for the new repository. Keep it URL-friendly: just letters, digits, and dashes; no whitespace. Character length is only limited by your OS. There is no database limit on the text field.
Inherit Rights From:	By default, your repo inherits access rights from the top-level Code Review repo ("project") named "All-projects". However, you can specify a different repo for more control.
Only serve as parent for other repos	Use this option if you wish to create a repo with specific access rights which other repos can inherit.

To Configure an Existing Repository

1. Click on a repository name.
2. Select **General**, **Branches**, **Access**, or **AccuRev Connector**.



General

The **General** menu allows you to specify basic repo properties, primarily associated with Gerrit Code Review features. The various fields are described in *Table 2* below.

Table 2. General Repo Settings

Setting	Description
Description	Enter a string that will appear in the "Repository Description" column of the repository lists.
Git Repository Options	**Ignore Case** -- When checked, sets `core.ignorecase=true` in the repository's **.git/config** file. **Ignore Symbolic Links** -- When checked, sets `core.symlinks=false` in the repository's **.git/config** file.

Table 2. General Repo Settings

Setting	Description
Gerrit Repository Options	**Submit Type** -- Options that allow you to control how to merge changes: **- Fast Forward Only** **- Merge if Necessary** **- Always Merge** **- Cherry Pick** For a discussion of these merge options, see the *Project Configuration* topic in the Gerrit Code Review documentation. **State** -- Determines the accessibility of the repository: **- Active** -- Users can view, pull from, clone, or push to this repository (assuming that they have the correct access rights). **- Read Only** -- Users can see and pull from this repository (assuming that they have read permission) but cannot modify or push to it. **- Hidden** -- Only Owners can see the repo. **Automatically Resolves Conflicts** -- By default, Code Review attempts merges only if there is no path conflict. Enabling this option forces Code Review to attempt the merge even if a path conflict occurs. **Require Change-Id in commit message** -- For a discussion of Change-Ids, see the *Change-Ids* topic in the Gerrit Code Review documentation. (**Note**: Does not apply to commits pushed directly to a branch or tag.)
Contributor Agreement	**Require Signed-off-by in commit message --** For a discussion of Signed-off-by lines, see the *Signed-off-by Lines* topic in the Gerrit Code Review documentation. **Require a valid contributor agreement to upload --** For a discussion of valid contributor agreements, see the *Contributor Agreements* topic in the Gerrit Code Review documentation.
Basic Permissions	**Enable Code Review** -- Indicates whether or not GerritCode Review is enabled for the current repository. By default, this field is checked. See *Enable/Disable Code Review* on page 52 for more information.
Clone Command	Lets you quickly copy the `git clone` command for the current repository specified using either the HTTP or SSH protocol.

Branches

The Branches menu displays the existing branches associated with this repo, and provides the ability to delete a branch or create a new one. See *Create Branches for a Repo* on page 34 for a description of how to use this section.

The labels in the table are relatively self-explanatory:

- **Branch Name**: See the Git documentation for **git-check-ref-format** for restrictions regarding Git branch names.
- **Revision**: This is typically either “HEAD”, or the name of another branch, or the SHA1 hash for a specific commit.

Access

The Access menu allows you to apply GitCentric group-based permissions (ACLs) on your repositories.

These ACLs are derived from Gerrit Code Review ACLs.

Procedural information for setting ACLs is provided at *Configure Access Rights (ACLs) for a Repo* on page 34.

Overview and detailed information about group-based ACLs is provided at:

- *GitCentric Group-Based ACLs* on page 10.
- The *Change-Ids* topic in the Gerrit Code Review documentation.

1. Most ACLs take **Allow**, **Deny**, or **Block**. values. However, **Label Verified** and **Label Code-Review** take numeric values which are displayed as reviewer options on the Code Review page.
2. Each permission provides an **Exclusive** checkbox, which grants an exclusive ref-level access control so that only members of a specific group can perform an operation on a repository/reference pair. See the *Access Controls* topic in the Gerrit Code Review documentation for more information.
3. The **Push** and **Push Annotated Tag** permissions also provide a **Force Push** option, which allows an existing branch to be deleted. Since a force push is effectively a delete immediately followed by a create, but performed atomically on the server and logged, this option also permits forced push updates to branches. Enabling this option allows existing commits to be discarded from a project history.

AccuRev Connector

Use the AccuRev Connector menu to:

- Define the mapping between branches in your Git repositories and mount points in your AccuRev streams. This procedure is described in *Map a Git Branch to an AccuRev Stream* on page 38.
- Implement AccuRev change packages. This procedure is described in *Enable and Use Change Packages* on page 43.

Table 3. AccuRev Connector Settings

Section	Field	Description
AccuRev Server Connection	**Server name:port**	The AccuRev server and port where the stream to be mapped exists.
	Username	The <*gc_user*> account that is a member of the group that is set to ASSIGN_USER_PRIVILEGE.
	Password	The password for <*gc_user*>.
Associate Issues With Commit	**Issue Tracking System**	If you are using AccuRev change packages, use this field to specify whether you are using AccuWork or a third-party product such as Rally as your issue tracking system.
	Regular Expression	If AccuRev uses an issue tracking system (either AccuWork, or a third-party system such as Rally), you can specify a regular expression here to enforce comment requirements for AccuRev change packages. Populating this field enables change package integration; leaving it blank disables it.

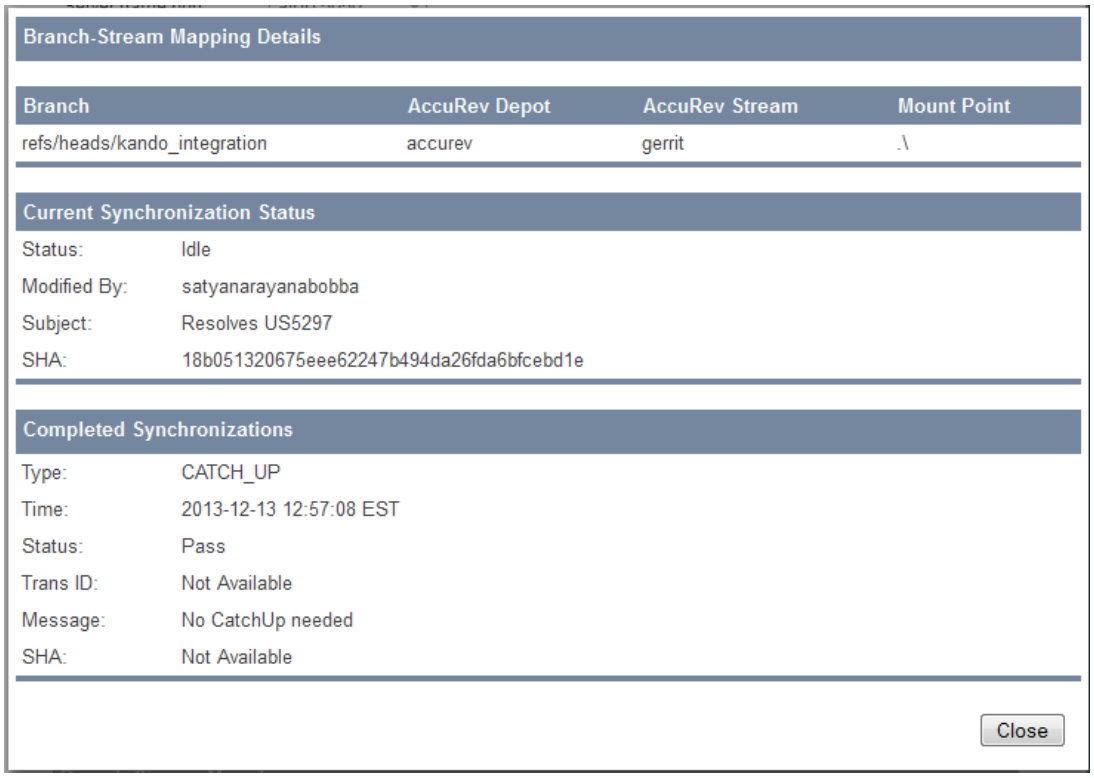
Table 3. AccuRev Connector Settings

Section	Field	Description
Branch-Stream Mappings	**Branch**	The branch that you want to map to an AccuRev stream.
	AccuRev Depot	The AccuRev depot containing the stream that the Git branch is mapped to.
	AccuRev Stream	The AccuRev stream to which the current branch is to be mapped.
	Mount Point	The directory in the mapped stream to which the branch is to be synchronized. Navigate to the mount point in the graphical tree and click it to populate the **Mount Point** field
	Initial Synchronization	This determines in which direction the first mapping will occur. If you are importing an existing repo into AccuRev, select **Commit Git content to Accurev**. If you are exporting the existing content of an AccuRev stream to a newly mapped repo, select **Commit Accurev content to Git**.

The informational line between the server and branch panels shows any existing mappings. The **Status** column shows *Active* whenever an import or export is occurring between the branch and the stream, and Idle when the synchronization is complete. Click the **View Details** button to display a real-time Status Monitor.

Status Monitor

The Status Monitor displays information from the GitCentric bridge for current and completed synchronizations for the selected branch-stream mapping. The Status Monitor updates every five seconds during an import or export operation.



Current Synchronization Status

The following table summarizes the values for the Current Synchronization Status fields.

Field	Description
Status	The status of the current synchronization. Values are: *Idle* -- the synchronization is complete *Syncing* -- the synchronization has just started *Syncing* | *Running for x seconds* -- the synchronization is active and has been running for the time shown.
Modified By	The user who pushed the commit.
Subject	The first line of the commit message entered by the user.
SHA	The SHA associated with the commit.

Completed Synchronizations

The following table summarizes the values for the Completed Synchronizations fields.

Field	Description
Type	The task type associated with the commit. See *Synchronization Type Values* on page 71 for details.
Time	The time the synchronization finished.
Status	Whether the synchronization task succeeded (*Pass*) or failed (*Fail*).
Trans ID	The AccuRev transaction associated with the commit.
Message	A message indicating that the synchronization passed or why it failed.
Subject	The first line of the commit message entered by the user.
SHA	The SHA associated with the synchronization.

The Status Monitor tracks phases per commit, so if you do a multi-commit push (such as a push on an existing repository to import commit history import into AccuRev), you will see these phases repeat over and over for each commit import. The most recent completed synchronization always appears at the top of the Complete Synchronizations section.

Synchronization Type Values

The following table summarizes the values for the Type field.

Table 4. Repo Status Entries

Task Type:	Phase:	Steps:
IncrementalImport (16 phases total)	Initialization	1 "step"
	doImport - Cleanup Workspace doImport - Parse Commit Messages doImport - Process Commit Diff	1 "step" 1 "step" n "diffs"
	setupPromote - processEmptyDirs setupPromote - processMoveAside setupPromote - checkEvilTwinForMove setupPromote - checkUndefunctFiles setupPromote - checkUndefunctDirs setupPromote - checkEvilTwinFiles setupPromote - undefunctElements setupPromote - coFiles setupPromote - processElinks setupPromote - popFiles setupPromote - moveFiles setupPromote - deleteFiles setupPromote - blobFiles setupPromote - addFiles setupPromote - keepFiles setupPromote - linkFiles setupPromote - mergeFiles setupPromote - chmodFiles setupPromote - removeOverlaps	1 "step" 1 "step" 1 "step" 1 "step" 1 "step" 1 "step" n "elements" 1 "step" n "elements" 1 "step" n "elements" 1 "step" n "blobs" 1 "step" 1 "step" 1 "step" 1 "step" 1 "step 1 "step"
	doImport - Promote files	1 "step"

Task Type:	Phase: (Continued)	Steps:
FullImport (19 phases total)	Initialization	1 "step"
	streamToCommit - Init streamToCommit - Process Elements streamToCommit - Finalize Commit	1 "step" n "elements" 1 "step"
	doImport - Cleanup Workspace doImport - Parse Commit Messages doImport - Process Commit Diff	1 "step" 1 "step" n "diffs"
	setupPromote - processEmptyDirs setupPromote - processMoveAside setupPromote - checkEvilTwinForMove setupPromote - checkUndefunctFiles setupPromote - checkUndefunctDirs setupPromote - checkEvilTwinFiles setupPromote - undefunctElements setupPromote - coFiles setupPromote - processElinks setupPromote - popFiles setupPromote - moveFiles setupPromote - deleteFiles setupPromote - blobFiles setupPromote - addFiles setupPromote - keepFiles setupPromote - linkFiles setupPromote - mergeFiles setupPromote - chmodFiles setupPromote - removeOverlaps	1 "step" 1 "step" 1 "step" 1 "step" 1 "step" 1 "step" n "elements" 1 "step" n "elements" 1 "step" n "elements" 1 "step" n "blobs" 1 "step" 1 "step" 1 "step" 1 "step" 1 "step" 1 "step"
	doImport - Promote files	1 "step"
FullExport (4 phases total)	Initialization	1 "step"
	streamToCommit - Init streamToCommit - Process Elements streamToCommit - Finalize Commit	1 "step" n "elements" 1 "step"
IncrementalExport or **CatchUpExport** (6 phases total)	exportSingle - Init	1 "step"
	doExport - getTransInfo doExport - getChangeSet doExport - generateFastImport doExport - gitFastImport	1 "step" 1 "step" n "changes" 1 "step"
	exportSingle - Check null commit	1 "step"

Support for Hooks

GitCentric supports both Git and Gerrit Code Review hooks. Borland recommends using Gerrit hooks when your environment requires hooks.

For more information about Gerrit Code Review hooks, see the Gerrit Code Review documentation, here:

https://gerrit-review.googlesource.com/Documentation/config-hooks.html

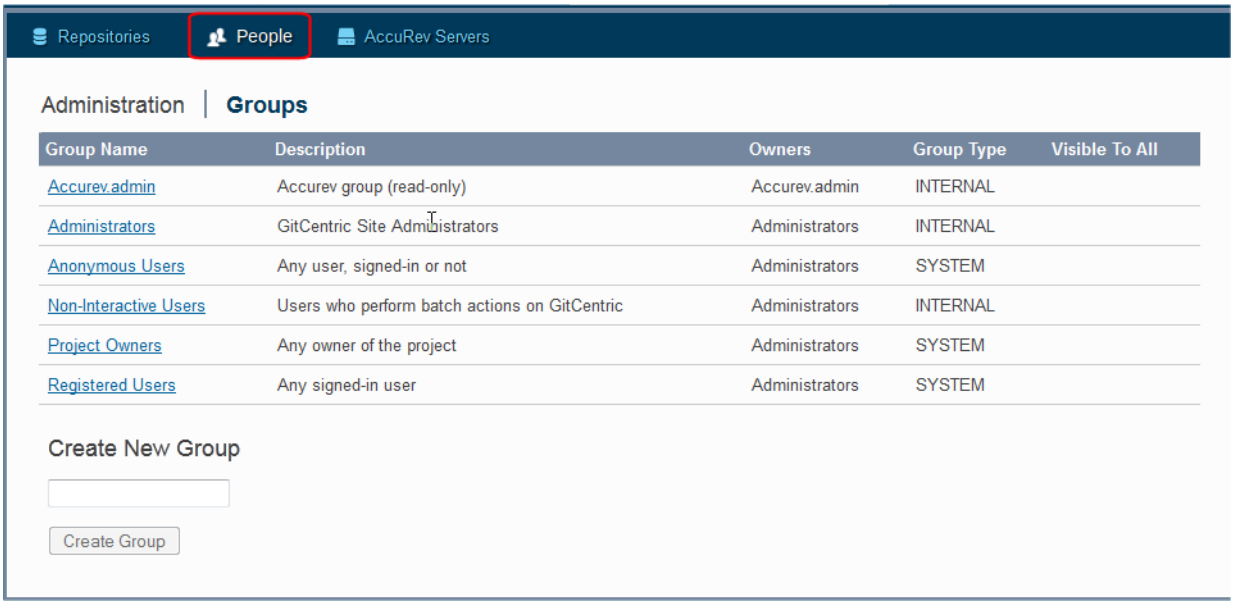
Migrating Existing Git Hooks

Two Gerrit hooks, ref-update and ref-updated, are installed to `<gc_home>/site/hooks` and are used to migrate any Git hooks you might have created in the `<gc_home>/site/git/<repoName>.git/hooks` directory.

Note that the ref-update and ref-updated hooks require Perl and expect to find it at `/usr/bin/perl`. If you have Perl in a different location, you will need to update these hooks with the correct path.

People

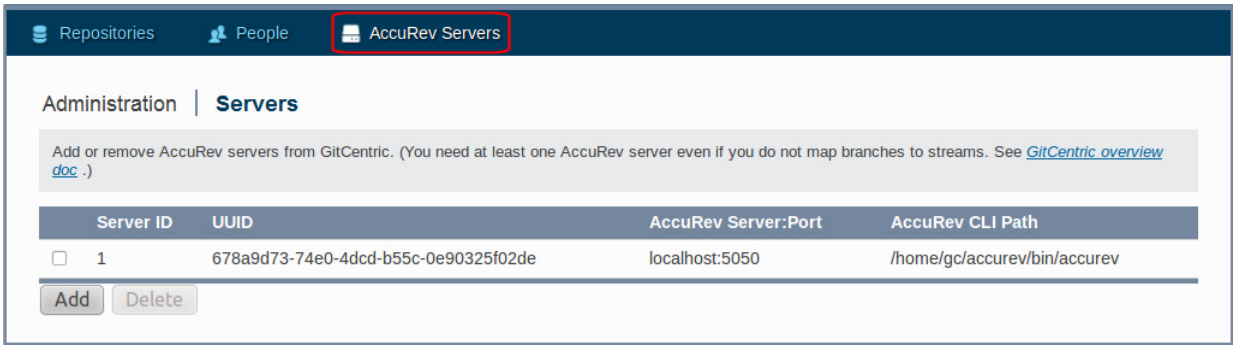
Group membership determines access rights in GitCentric. Use the Groups page to list existing groups, create a new group, and to access details about an existing group.



See *Manage GitCentric Groups* on page 49 for information about using this page.

AccuRev Servers

The AccuRev Servers panel displays information about configured AccuRev Servers and allows you to add or remove them from GitCentric.



When you click the **Add** button, the Servers page expands to display the fields you use to add an AccuRev server to GitCentric. These fields are summarized in the following table:

Server Field:	Description
AccuRev Server	The host of the AccuRev server that GitCentric associates with one or more repos. This can be an IP address or the name of the server. Note that if you are connecting to a remote GitCentric server and you specify `localhost`, this indicates an AccuRev server on the GitCentric host, not your local machine.
Port	This is typically "`5050`" for most AccuRev servers, but the AccuRev administrator may have changed it to something different.
AccuRev CLI Path	The path to the AccuRev *client* executable installed on the GitCentric server (not the remote AccuRev server). For example: `/opt/accurev/bin/accurev`. For more information, see *A Note about the CLI Path Setting* on page 74.

See *Add an AccuRev Server* on page 46 for information about using this panel to add a server.

A Note about the CLI Path Setting

When logging into GitCentric for the first time, or when configuring an AccuRev server to work with GitCentric, you are prompted to enter a value for the "CLI Path". This is the full path to the AccuRev client executable on the GitCentric server that should be used to communicate with the AccuRev server. For example:

<ac_home>`/bin/accurev`

where *<ac_home>* is the actual install directory, such as `/opt/accurev`.

The GitCentric server requires either a master AccuRev server or a compatible AccuRev client installed locally. In the case where you are using AccuRev servers of different revision levels, you must have a compatible client installed on the GitCentric server machine for each version of the AccuRev server you are using, and each must be configured with GitCentric using its specific path. For example, if you need to work with two AccuRev servers -- one running version 5.6 and one running version 5.7 -- you would need two AccuRev clients installed on the GitCentric server: one 5.6 client to work with the 5.6 server, and a separate 5.7 client to work with the 5.7 server.

A. The kandoMaintain Utility

The **kandoMaintain** utility provides the ability to perform occasional administrative operations under the guidance of an AccuRev Support Services representative. It is similar to the AccuRev **maintain** utility (see "The 'maintain' Utility" chapter of the AccuRev *Administrator's Guide*).

Using kandoMaintain

The **kandoMaintain** utility is located in the **bin** directory where you installed GitCentric. To use **kandoMaintain**, the AccuRev database server must be running, but we recommend that you stop the GitCentric server, which you do by shutting down Tomcat, for example:

```
cd <ac_home>/webUI/tomcat/bin
./shutdown.sh
```

All **kandoMaintain** commands except **help** require a database administrator user and password. You can enter the user name and password on the command line with the **-u** and **-P** options, respectively. If you do not supply these, **kandoMaintain** will prompt you for them. You can also provide either or both of these values in a text file and provide a path to the file (see the `-Fc <path>` option).

> **Note:** Providing the database password in a script or in a configuration file can compromise system security. Make sure that such files are protected and not generally accessible.

The commands to make, remove, upgrade, back up, or restore a database (**mkdb**, **rmdb**, **upgradedb**, **backupdbs**, **restoredbs**) prompt you to confirm the operation before proceeding. This confirmation prompt can be overridden with the `-y` option.

Backup and Restore

The **kandoMaintain** commands to back up and restore the GitCentric database (**backupdbs** and **restoredbs**) are described in this appendix. For complete information about the backup and restore process, see *Appendix: B Backup and Restore*.

kandoMaintain Command Reference

Usage: `kandoMaintain [COMMAND] [OPTIONS]`

```
kandoMaintain [ mkdb | rmdb | upgradedb | backupdbs | restoredbs | mvrepos |
           testconn | lsconfig | addconfig | rmconfig | checksync | help ]

           [ -u <dbadmin> -P <dbpass> -c <url> | -Fc <path> ]
```

Commands

mkdb

```
kandoMaintain mkdb -u <db_admin> -P <dbpass>
```

Creates a GitCentric database with the latest schema.

Valid options (see below): `-u, -P, -c, -Fc, -y`

rmdb

`kandoMaintain rmdb -u <db_admin> -P <dbpass>`

Removes a GitCentric database

Valid options (see below): `-u, -P, -c, -Fc, -y`

upgradedb

`kandoMaintain upgradedb -u <db_admin> -P <dbpass>`

Upgrades a GitCentric database to the latest schema.

Valid options (see below): `-u, -P, -c, -Fc, -y, -I`

backupdbs

`kandoMaintain backupdbs -u <db_admin> -P <dbpass>`

Creates a backup of the GitCentric database and writes the backup file to the directory from which the `backupdbs` command was run; the backup file can be moved to any location after that. Each backup file is given a unique file name, such as **kando_*<timestamp>*.backup**, where the *timestamp* is in the format of *yyy-mm-dd_hh:mm:ss:ms* (2012-03-16_15:51:11.487, for example).

The `backupdbs` command uses PostgreSQL `pg_dump`, which is located in the PostgreSQL **/bin** directory. Before running `backupdbs`, make sure that the PostgreSQL **/bin** directory is specified in your system path.

Valid options (see below): `-u, -P, -c, -Fc, -y`

Note: Micro Focus recommends that you use `full_backup.sh` and `full_restore.sh` commands to back up and restore GitCentric. See *Appendix: B Backup and Restore* for more information.

restoredbs

`kandoMaintain restoredbs -u <db_admin> -P <dbpass> -Fb <path>`

Restores the backup of the GitCentric database created by the `backupdbs` command. Use the `<path>` argument to specify the location of the backup file. By default, the backup file is written to the same directory from which the `backupdbs` command was run, but it can be moved anywhere.

Valid options (see below): `-u, -P, -c, -Fb, -Fc, -y`

mvrepos

`kandoMaintain mvrepos -u <db_admin> -P <dbpass> -r <destination_path>`

Move any Kando 2012.1 repositories from their old storage area to the new GitCentric storage area at `<destination_path>` (typically `<gc_home>/site/git`) after an upgrade installation. `mvrepos` identifies the old storage area by querying the database.

`mvrepos` checks that the target path exists and is writable. `mvrepos` only looks at acrepository records that are not hidden (deleted). If there is a null or empty path in any of the acrepository records, no moves will be attempted. Any of the above conditions will return error messages as well as a return status of 1. During the individual moves and database updates, if a single move fails, the others will still be attempted. There will be error messages for each move that fails and the overall return status will be 1. Therefore, you can rerun this command after fixing a failure condition, so that eventually all repos are moved. If all moves are successful, no message is displayed and a 0 is returned for status.

Valid options (see below): `-u`, `-P`, `-c`, `-Fc`, `-y`, `-r`

testconn

`kandoMaintain testconn`

Tests the Java Database Connectivity (JDBC) connection to the postgres instance (does not test for the existence of a GitCentric database.

Valid options (see below): `-u`, `-P`, `-c`, `-Fc`

lsconfig

`kandoMaintain lsconfig`

Lists the entries from the GitCentric database configuration table. Use `-n` to list a specific configuration parameter. If `-n` is not specified, all configuration parameters are displayed.

Valid options (see below): `-u`, `-P`, `-c`, `-Fc`, `-n`

addconfig

`kandoMaintain addconfig -n <name> -v <value>`

Adds a configuration parameter to the GitCentric database configuration table. Returns an error if you specify a name that already exists with `-n <name>.`

Valid options (see below): `-u`, `-P`, `-c`, `-Fc`, `-n`, `-v`, `-m` (`-n` and `-v` are required)

rmconfig

`kandoMaintain rmconfig -n <name>`

Deletes a configuration parameter from the GitCentric database configuration table. (**Note:** no error is generated if the specific configuration name does not exist.)

Valid options (see below): `-u`, `-P`, `-c`, `-Fc`, `-n` (`-n` is required)

checksync

`kandoMaintain checksync`

Verifies that the latest branch-stream synchronization for the specified branch is correct. For the latest shamap entry for the specified branch-stream mapping, `checksync` returns the sha, stream as string, transaction, timestamp, and operation fields, and identifies any discrepancies between:

- The current branch sha and that shamap entry
- The current stream high water mark (HWM, which is based on the transaction number of the last change made to the content of an AccuRev stream) and that sha map entry

Use the `--diff` option to compare the contents of each directory, file, and link on Git and AccuRev.

Valid options (see below): `-Fc`, `-r`, `-b`, `--diff` (`-r` and `-b` are required)

help

`kandoMaintain help`

Displays usage information.

Valid options (see below): None.

Options

Connection Options

`-u <dbadmin>` -- The database administration user. This is the same user name as specified during installation (default is “postgres”). If this is required by the command you are using but you do not include it, kandoMaintain prompts you for it.

`-P <dbpass>` -- The database administration password. This is the same user password as specified during installation. If this is required by the command you are using but you do not include it, kandoMaintain prompts you for it. If you write scripts that incorporate this option, be sure to secure the file against unauthorized access.

`-c <dbconnectstring>` -- The database connection string: `jdbc:postgresql://<host>:<port>/<dbname>`. If you do not specify `-c` for a command that requires it, kandoMaintain assumes the default value "`jdbc:postgresql://localhost:5075/kando`". If you do specify `-c`, then "`jdbc:postgresql://`" at a minimum is required. Partial connection strings will have default values automatically supplied. For example, if you specify "`jdbc:postgresql://`", kandoMaintain assumes a `<host>` of “localhost”, a `<port>` value of “5075” and a `<dbname>` of “kando”.

`-Fc <path>` -- The path to a text file (such as `<gc_home>/dbsettings.conf`) containing values for three of the parameters used by kandoMaintain. The contents of this file should include:

```
DB_USER=<dbadmin> (eliminates need for "-u" on the command line)
DB_PASS=<dbpassword> (eliminates need for "-P" on the command line)
DB_CONNECT=<dbconnectstring> (eliminates need for "-c" on the command line)
```

For example:

```
DB_USER=postgres
DB_PASS=postgres
DB_CONNECT =dbc:postgresql://localhost:5075/kando
```

Note: The settings and the values accepted by this configuration file can change without notice from release to release.

Other Options

`-Fb <path>` -- The path to the backup file of the GitCentric database created by the `backupdbs` command. Used by `restoredbs`.

`-n <name>` -- Configuration name (the `param_name` field in the GitCentric database configuration table). See *GitCentric Bridge Configuration Settings* below.

`-v <value>` -- Configuration value (the `param_value` field in the GitCentric database configuration table). See *GitCentric Bridge Configuration Settings* below.

`-m '<comment>' --` Configuration comments (the `comments` field in the GitCentric database configuration table). (Double or single quotes are required if the comment includes spaces.)

`-r <repository_path>` -- The path to the GitCentric repository. For `mvrepos`, the path to the repository to which the command should move existing repositories after an upgrade installation. For `checksync`, the path to the repository whose synchronization status you are checking.

`-b <branch>` -- The name of the branch in the GitCentric repository whose synchronization status you are checking with the `checksync` command.

`-y` -- Suppress confirmation prompting for `mkdb`, `rmdb`, `mvrepos`, or `upgradedb`.

`-I` -- When executing an `upgradedb` command, initialize a new database if one does not exist.

`--diff` -- For `checksync`, compares each directory, file, and link between the Git branch and AccuRev stream whose synchronization status you are checking and reports any discrepancies.

GitCentric Bridge Configuration Settings

Here are the configuration values that can be set with

```
kandoMaintain addconfig -n <name> -v <value>
```

The bridge is the webapp `<tomcat_home>/webapps/kandoBridge` that keeps your Git repositories and AccuRev streams in sync.

Table 1. GitCentric Systemwide Configuration Settings

Name	Value	Comments
`AC_GIT_BIN`	Full path and name of Git executable.	Typically set by the GitCentric installer and should not need to be modified. If this value is not defined, GitCentric uses the PATH to find the Git executable.
`ACCUREV_COMMAND_LOGFILE`	Full path and name of a log file.	DEBUG ONLY. Use only when directed by AccuRev Support. This log shows each AccuRev command that gets executed by the bridge. **Note:** For Code Review and UI logging, see: `<gc_home>/site/etc/gerrit.config`
`PRESERVE_TEMP_FILES`	[Y | y]	DEBUG ONLY. Use only when directed by AccuRev Support. "Y" specifies to not delete temporary files (from xmlpromote, fast-import, misc. messages, etc.) that are generated in `$CATALINA_HOME/temp`.
`AC_BRIDGEAPI_SECURITY_POLICY`	AllowAnyHost	DEBUG. By default GitCentric recognizes trigger events only from trusted servers. Use this setting to troubleshoot if a trusted server is having difficulty communicating with GitCentric. (Case sensitive.)
`GIT_RENAME_THRESHOLD`	[0 - 100]	Value for matching the Git rename detection threshold, so that a file delete/replace can be identified as a rename operation, allowing AccuRev to track the history of a renamed file. The default value is 50 which is the same as the default Git setting. Only adjust this value if renames are not identified accurately.
`kando.bridgeURL`	URL of the GitCentric bridge	Allows you to set the URL of the GitCentric bridge; used when the GitCentric server and AccuRev server are on different hosts. For example: `http://<host name>:8080/KandoBridge` where `<host name>` is the name or IP address of the GitCentric server host.

Examples

1. Test the JDBC connection to the PostgreSQL database instance for GitCentric, specifying the DB user account but having kandoMaintain prompt for the DB password:

   ```
   kandoMaintain testconn -u postgres
   ```

   kandoMaintain prompts you to enter the database admin password, and uses the default connection string "`jdbc:postgresql://localhost:5075/kando`" since you did not specify the `-c` option.

2. List all the entries in the GitCentric database configuration table. Specify the database username, password, and connection string in an external file called db_vals.txt which exists in a secure directory named "securefiles".

   ```
   kandoMaintain lsconfig -Fc ./securefiles/db_vals.txt
   ```

3. Remove the configuration value that points to the instance of Git used by GitCentric, specifying the DB user account and DB password:

   ```
   kandoMaintain rmconfig -u postgres -P DBpass2901 -n AC_GIT_BIN
   ```

4. Add a new configuration value that points to the instance of Git used by GitCentric, specifying the DB user account but having kandoMaintain prompt for the DB password:

   ```
   kandoMaintain addconfig -u postgres -n AC_GIT_BIN -v '/usr/local/bin/git'
   ```

5. After upgrading GitCentric, upgrade the GitCentric database with the latest schema. Suppress the confirmation prompt, and specify the database username, password, and connection string in an external file called db_vals.txt which exists in a secure directory named "securefiles".

   ```
   kandoMaintain upgradedb -y -Fc ./securefiles/db_vals.txt
   ```

B. Backup and Restore

This appendix describes the commands and processes you can use to back up and restore GitCentric.

Commands for Backup and Restore

GitCentric provides two sets of commands for backing up and restoring GitCentric, summarized in the following table:

Table 1. Summary of Backup and Restore Commands

Command	Description	When to Use
`full_backup.sh` `full_restore.sh`	The `full_backup.sh` command backs up the GitCentric databases, your Git repositories, and your configuration files. Likewise, `full_restore.sh` copies all required files to the new location.	Use `full_backup.sh` and `full_restore.sh` any time you wish to completely back up GitCentric. You should also use these commands when migrating GitCentric from one server to another.
`backupdbs` `restoredbs`	The low-level `backupdbs` and `restoredbs` commands, as the names imply, back up and restore *only* the GitCentric databases. Other important data, like your Git repositories and configuration files are *not* backed up by the `backupdbs` command. `backupdbs` and `restoredbs` are `kandoMaintain` commands. See *Appendix A: The kandoMaintain Utility* for more information.	You might wish to use these commands if your current backup strategy includes a third-party tool for backing up your repositories. **Note**: If you are currently using `backupdbs` and `restoredbs` to back up GitCentric, you can continue to do so.

Micro Focus recommends that you use the `full_backup.sh` and `full_restore.sh` commands to back up and restore GitCentric as described in the remainder of this appendix.

Best Practices

Micro Focus recommends that you back up GitCentric on a daily basis -- every night, for example. You can back up GitCentric at any time, but Micro Focus recommends that you choose a time of light system usage.

Tip: Consider creating a cron job for GitCentric backups. For examples of cron jobs, see *Set Up Gerrit Garbage Collection* on page 55.

Backing Up GitCentric

This section describes the background information and procedures you use to back up GitCentric.

What Gets Backed Up?

The `full_backup.sh` command backs up:

- all Git repositories
- all GitCentric databases
- all configuration settings -- database connection, Gerrit configuration (**site/etc**), Gerrit hooks (**site/hooks**), SSH keys used for replication (**site/ssh**)

All files are backed up to a .tar file that is created at the location from which you run the `full_backup.sh` command. The file is named **kando_backup_site_etc_<*yyyymmdd*>.tar**, where **<*yyyymmdd*>** is the year, month, and day.

You can move the backup file to -- and restore it from -- a different location if you choose.

What are the GitCentric Databases?

The GitCentric databases, "kando" and "gcreviewdb", are PostgreSQL databases installed on the GitCentric server. These databases contain all AccuRev stream-Git branch mappings, as well as a history of all import and export activity. Stream-branch mappings are recorded using AccuRev transaction-Git commit pairs. For more information on the GitCentric databases, see *Basic Architecture* on page 3.

> **Note:** The GitCentric databases are entirely separate from the AccuRev database; the procedures described in this appendix apply only to the GitCentric databases; they have no effect on the AccuRev database.

How to Back Up GitCentric

1. Stop the GitCentric server using `shutdown.sh`:

   ```
   <ac_home>/WebUI/tomcat/bin/shutdown.sh
   ```

2. Run `full_backup.sh -l <path to gc_home>`
   where
   `<path to gc_home>` is the full path to the GitCentric home directory you want to back up.

   For example:

   ```
   ./full_backup.sh -l /home/gitcentric/AccuRevGitCentric
   ```

   GitCentric displays a completion message, including the name of the .tar file, if the backup was successful; otherwise, it displays errors.

3. Start the GitCentric server using `startup.sh`:

   ```
   <ac_home>/WebUI/tomcat/bin/startup.sh
   ```

   Note: Make sure that the user starting Tomcat has write access to the **logs**, **temp**, **webapps**, and **work** directories in **<*ac_home*>/WebUI/tomcat**. This user should have read access to all other Tomcat directories and files.

Restoring GitCentric

This section describes the background information and procedure you use to restore GitCentric.

Caution: Restore Overwrites Existing GitCentric Installations

The `full_restore.sh` command completely overwrites an existing GitCentric installation. If you are restoring to an existing GitCentric installation, make sure there is no data under the GitCentric home directory that needs to be saved.

Prerequisites

In order to successfully run the `full_restore.sh` command:

- you must have an existing GitCentric installation to use as the target for the restore operation. This GitCentric installation can be empty.
- ***<ac_home>*/jre/bin** must be in your system's search path (set using the $PATH environment variable)
- ***<ac_home>*/postgresql/bin** must be in your system's search path (set using the $PATH environment variable)

What Gets Restored?

The `full_restore.sh` command restores all the files and databases in the **kando_backup_site_etc_*<yyyymmdd>*.tar** created by the `full_backup.sh` command. See *What Gets Backed Up?* on page 84 for more information.

How to Restore GitCentric

To restore GitCentric:

1. Stop the GitCentric server using `shutdown.sh`:

   ```
   <ac_home>/WebUI/tomcat/bin/shutdown.sh
   ```

2. Run `full_restore.sh -l <path to gc_home> <backup.tar file>`
   where
   <path to gc_home> is the full path to the GitCentric home directory to which you want to restore
   <backup.tar file> is the name of the .tar file created by `full_backup.sh`.

   For example:

   ```
   ./full_restore.sh -l /home/gitcentric/AccuRevGitCentric
   kando_backup_site_etc_<20150213>.tar
   ```

3. **Important:** Review the `full_restore.sh` output and perform any required actions. See *Restore Scenarios* on page 87 for more information on restore situations you may encounter.
4. Restart Tomcat:

   ```
   <ac_home>/WebUI/tomcat/bin/startup.sh
   ```

Note: Make sure that the user starting Tomcat has write access to the `logs`, `temp`, `webapps`, and `work` directories in *<ac_home>*`/WebUI/tomcat`. This user should have read access to all other Tomcat directories and files.

Next Steps

If your GitCentric restore was successful, you might need to perform one or more of the following tasks depending on whether or not you restored GitCentric to a new location or to a new machine. In addition, you might need to merge Gerrit configuration files.

If You Restored to a New Location

If you restored GitCentric to a new location on the existing machine, you need to:

1. Modify `docBase` in **<*ac_home*>/WebUI/tomcat/conf/Catalina/<*hostname*>/gitcentric.xml** and **kandoBridge.xml** to point to the new GitCentric location.
2. Compare Gerrit configuration files in **kando_backup_site_etc_<*yyyymmdd*>.tar** and merge any changes required to support your GitCentric installation to the new **../site/etc**.
3. Restart Tomcat:

   <ac_home>`/WebUI/tomcat/bin/shutdown.sh`

   <ac_home>`/WebUI/tomcat/bin/startup.sh`

If You Restored to a New Machine

If you restored GitCentric to a new machine, as you might in a migration scenario, for example, you need to:

1. Set up Tomcat on the new machine with:

   <*ac_home*>/WebUI/tomcat/conf/Catalina/<*hostname*>/gitcentric.xml

   and

   <*ac_home*>/WebUI/tomcat/conf/Catalina/<*hostname*>/kandoBridge.xml
2. Compare Gerrit configuration files in **kando_backup_site_etc_<*yyyymmdd*>.tar** and merge any changes required to support your GitCentric installation to the new **../site/etc**.
3. Restart Tomcat to adjust AccuRev settings:

   <ac_home>`/WebUI/tomcat/bin/shutdown.sh`

   <ac_home>`/WebUI/tomcat/bin/startup.sh`
4. Use PostgreSQL `psql` or `pgadmin` tools to edit the Kando database "acserver" table to correct the server client path for each AccuRev server.

If You Restored to the Same Location

If you restored to the same location, simply unpack the **kando_backup_site_etc_<*yyyymmdd*>.tar** file by running:

```
tar xvf kando_backup_site_etc_<yyyymmdd>.tar
```

Restore Scenarios

In typical usage, GitCentric displays a message like the following for each Git branch that is successfully restored:

```
Processing branch refs/heads/master in repository /sandbox/llowry/tmp/ws/git/
Repo2.git/

==============================

Branch is in sync with the GitCentric database.
```

When the restore is successful, no action is required on the part of the GitCentric administrator. The remainder of this section describes other scenarios that require the attention of the GitCentric administrator.

Overflow

An *overflow* condition occurs when the GitCentric database is backed up before the Git repositories are copied. In this event, it is possible that the GitCentric database and the Git repository copies are out of synch -- specifically, the GitCentric database probably does not have a complete set of AccuRev transaction-Git commit pairs reflecting the latest commits in the Git repositories.

When this happens, the GitCentric restore:

- identifies the latest mapped Git commit of which it is aware and resets the head there.
- creates a new branch that contains the commits that are not recorded in the GitCentric database. The new branch is named **kando_backup_overflow_<*name*>** where **<*name*>** is original branch name.
- Writes a message to output like the following:

```
Processing branch refs/heads/maint in repository /sandbox/llowry/tmp/ws/git/
Repo2.git/

==============================

Creating kando_backup_overflow_maint; review divergence with maint and merge as
necessary.
```

In the case of an overflow condition, it is up to the GitCentric administrator to determine whether or not these changes need to be merged into the original branch and pushed to the Git repository. If the changes represented by the unrecorded commits are already in AccuRev, GitCentric will automatically export them to the restored repository after you start the GitCentric server. If the missing changes are not in AccuRev, the user can clone, then fast forward-only merge from the overflow branch to the named branch, and push.

Rollback

A *rollback* condition can occur when there is a large interval of time between when the copy of the Git repositories was created and the GitCentric database backup was performed. In this event, the GitCentric database will have recorded Git commits which are not in the restored repositories.

When the GitCentric restore encounters a rollback condition, it writes a message to output like the following:

```
Processing branch refs/heads/hotfix in repository /sandbox/llowry/tmp/ws/git/
Repo1.git/

==============================

Rolling back mapped transactions that are higher than 61.
```

In this example, the GitCentric administrator needs to be aware that any AccuRev transaction-Git commit pair greater than transaction number 61 is not known to the Git repositories; GitCentric has rolled back

database transactions to 61 -- the last value known to the Git repositories. Users that have any missing changes in their clone should push them again.

Missing Branch in the Repository Copy

If GitCentric restore cannot locate a branch, it writes a message to output like the following:

```
Processing branch refs/heads/spec in repository /sandbox/llowry/tmp/ws/git/
Repo1.git/

==============================

Branch could not be found. Make sure the repository is at the expected location with
the appropriate branch.
```

In this case, the GitCentric administrator needs to ensure that the copies of all Git repositories have been copied to their original location (that is, the location at the time the GitCentric database backup was created) and that all expected branches are present in those repositories. You may need to delete and recreate the branch mapping in GitCentric.

C. Command-Line Reference

This chapter provides a detailed description of the CLI (Command Line Interface) commands available to GitCentric users. Although they can be used interactively, these commands are intended to be used in administrative scripts.

Basic Syntax

The GitCentric CLI commands are executed as SSH remote commands, which require that you have an SSH client installed. The syntax for a GitCentric CLI command executed from SSH is:

```
ssh -p <port> <user>@<host> gitcentric [<command>] [[<arg>]...]] [--help] [--]
```

where:

<port> is the GitCentric port number, typically `29418.`

<user>@<host> is the SSH user account used with GitCentric (for example `myusername@localhost`).

<command> may be one of the following:

- `config-branch`
- `config-repo`
- `delete-repo`
- `ls-repo`

<arg> is one or more arguments to pass to *<command>*. These vary based on command.

`--help, -h` displays the on-line help for the current command. **Note:** Specifying --help with no other arguments (for example, `gitcentric --help`) returns a list of all available sub-commands.

`--` specifies that the rest of the command line is to be considered arguments, not options. For example, if you need to pass a repo name that starts with "-" you would precede it with "--". Note that after using "--", you cannot pass any further options on that command line.

```
...config-repo Repo1 Repo2 -- -Repo3 Repo4 -Repo5
```

Spaces and Quoting

You must use quotes to include a space in arguments for a GitCentric SSH command.

In addition, Windows requires that you nest single quotes within double-quotes. In the example below, highlighted in bold, double-quotes are provided for the Windows shell, and single-quotes are nested within the double-quotes for the SSH command-line parser.

```
C:\source\TestProject>ssh -p 29418 mylogin@localhost gitcentric
config-branch --branch refs/heads/master -p 1 -e 3 -s 2
-d "'xxxx yyyy'" --initialSync git TestProject
```

config-branch — configure a branch or ref

Usage

```
gitcentric config-branch
[--branch <ref> | -b <ref>] [--clear] [--noNotify] [--description <description>]
[--initialSync <direction>] [--mountElement <mount_elem>] [--stream <stream>]
[--help] [<reponame>...] [--children-of <reponame>] [--]
```

Description

The `gitcentric config_branch` command enables you to configure Git branches and refs for use with GitCentric.

Specify one or more repositories to configure with arguments `<repo_1>` through `<repo_n>`, or specify all the children of a parent repository with `--children-of <reponame>`.

> **Note:** You must use the `config-repo` command before you can map a repo's branches to AccuRev streams.

Options

`--branch <ref>, -b <ref>`
: (Required.) Specify the branch or the ref to configure. If you have specified multiple repos, `config-branch` displays an error when it encounters a repo where the branch does not exist, but proceeds for those repos where the branch does exist. This option accepts branch specifications in either long form ("refs/heads/branch_XYZ") or short form ("branch_XYZ"), but saves the information in long form.

`--children-of <reponame>`
: Configure all the immediate children of the specified parent repo. (Children of children will not be configured.) This option is not intended for initial configuration, where you are unlikely to want to map multiple branches all to the same element in the same AccuRev stream. Rather, it is intended for maintenance, where you want to change the same configuration setting for many repos with a single operation. **Note:** If you specify repos in the argument list and with the --children-of option, the command operates on ALL repos.

`--clear`
: Remove the branch mapping.

`--description <description>, -d <description>`
: A text string describing this branch mapping. (See *Spaces and Quoting* on page 89 for more information.)

`--help, -h`
: Display the help for this command.

`--initialSync <direction>`
: Specify the direction which should be used by the first synchronization when mapping between a Git repo and an AccuRev stream. Valid values are case-insensitive and include:

  - `accurev` -- (Default) Use this option if you are creating a new Git repo and will be populating it by mapping its branch to an existing AccuRev stream.
  - `git` -- Use this option if you are you are mapping an existing Git branch to an unpopulated directory in an AccuRev stream.

`--mountElement <mount_elem>, -e <mount_elem>`

The name or numeric ID of the AccuRev directory element that the Git branch is mapped to. If the mount element is unspecified, GitCentric uses the forward slash (/) as the default root mount point. If you wish to specify the root mount point manually, you must use the forward slash (/); you cannot use the backward slash (\) or period (.).

`--noNotify`

Do not notify the GitCentric bridge of the configuration change. This is useful if you are making a series of configuration changes and do not want to reconfigure the bridge until they are all done. You do this by omitting `--noNotify` on the last command. If you forget to omit `--noNotify` on the last command, you can force the commit by either executing a `config-repo` or `config-branch` command, or by restarting the bridge (such as through the Tomcat admin console).

`--stream, -s <stream>`

The name of the AccuRev stream to map the branch to. Note: stream IDs are not accepted.

`--`

"End-of-options" marker. See *Basic Syntax* on page 89 for more details.

Examples

Initial configuration example: Map a branch named "branch_XYZ" which exists in a repository named "new_repo2" to the mount point directory "added_folder" which exists in an AccuRev stream named "agc_domestic".

```
>ssh -p 29418 mylogin@myGCserver gitcentric config-branch -b branch_XYZ
-s agc_domestic -e /./agc_devel_folder1/added_folder new_repo2
```

Maintenance example: Change the configuration for a repository named "new_repo2" as well as all the children of parent-only repository "test_parent", to clear ALL the existing mappings between branches and streams.

```
>ssh -p 29418 mylogin@myGCserver gitcentric config-branch  --clear
--children-of test_parent_1 new_repo2
```

Access

Any member of the Project-Owners or Administrators groups.

See Also

Basic Syntax on page 89, *config-repo*, *ls-repo*

config-repo — configure a repo

Usage

```
gitcentric config-repo [--server <server>:<port>] [--noNotify] [--clear]
[--useAccuworkKey | --useThirdPartyKey] [--cpkPattern <regex>]
[--user <user>] [--pw <password>][--help] [--children-of <reponame>]
[<reponame>...] [--]
```

Description

The `gitcentric config_repo` command enables you to configure Git repositories for use with GitCentric.

Specify one or more repositories to configure with `<reponame>...`.

You can use the `--useAccuworkKey` or `--useThirdPartyKey` options to configure a repo for an AccuRev server that uses either AccuWork or a third-party issue tracking system. For more information, see *Enable and Use Change Packages* on page 43.

If your site does use change packages, the `--cpkPattern` option can be used to specify a Regular Expression setting that determines what your users need to put in their Git commit comments so that AccuRev knows what issue the change applies to. A discussion of Java regular expressions is beyond the scope of this document, but here is an example:

> `Resolved\s+([\d,]+)` -- *This format would be adequate for simple environments where the comment always starts with "Resolved", followed by white space, followed by any number of issues, separated by commas:*
>
> Resolved 12576
> Resolved 34, 149, 11057, 686

Note: White space in regular expressions (and any other CLI argument) needs to be quoted (see *Spaces and Quoting* on page 89), and special characters may require escape characters or quoting. For example, for the above regular expression to be passed correctly from the command line, you would need to construct the option as follows: `--cpkPattern "Resolved'\'s+['\'d,]+)"`

Use a search engine and search for "Java regular expressions" for more information about these formats.

Options

`--children-of <reponame>`
: Configure all the children of the specified parent repo.

`--clear`
: Clear out all GitCentric configuration for this repository, including ALL branch configuration.

`--cpkPattern, -c <regex>`
: Specify an AccuRev change package regular expression that determines what your users need to put in their Commit comments so that AccuRev knows what issue the change applies to.

`--help, -h`
: Display the help for this command.

`--noNotify`
: Do not notify the GitCentric bridge of the configuration change. This is useful if you are making a series of configuration changes and do not want commit them until they are all done, by omitting `--noNotify` on the last command. If you forget to omit `--noNotify` on the last command, you can

force the commit by either executing a `config-repo` or `config-branch` command, or by restarting the bridge (such as through the Tomcat admin console).

`--pw, -p` *`<password>`*
: The AccuRev login password for *`<user>`*. If omitted then an empty password is used to log in to AccuRev.

`--server, -S` *`<server>:<port>`*
: Specify the IP address and the port to the AccuRev server that this repository is mapped to.

`--useAccuworkKey, -a,`
`--useThirdPartyKey, -3,`
: Use Accuwork issue numbers for change packages, or use third party ITS keys instead of AccuWork issue numbers for change packages. These options are mutually exclusive.

`--user, -u` *`<user>`*
: The AccuRev login account to use for this repository.

`--`
: "End-of-options" marker. See *Basic Syntax* on page 89 for more details.

Examples

Initial configuration example: Configure a repo named "proj_3000" to be associated with the AccuRev server at "myACServer:5050", which uses ASSIGN_PRIVILEGE_USER "agc_sync" with a password of "sync001":

```
> >ssh -p 29418 mylogin@myGCserver gitcentric config-repo proj_3000
    -S myACServer:5050 -u agc_sync -p sync001
```

Maintenance example: Change repos "proj_3000" and "proj_4000", as well as all of the immediate children of parent-only repo "off_shore_parent" to a new server (which uses the same ASSIGN_USER_PRIVILEGE user and password).

```
> >ssh -p 29418 mylogin@myGCserver gitcentric config-repo proj_3000 proj_4000
    --children-of off_shore_parent -S myACServer:5050 -u agc_sync -p sync001
```

Access

Any member of the Project-Owners or Administrators groups.

See Also

Basic Syntax on page 89, *delete-repo*, *config-branch*, *ls-repo*

delete-repo — delete repository

Usage

```
gitcentric delete-repo <reponame> [--force] [--noNotify] --yes-really-delete
[--help(-h)] [--]
```

Description

The `gitcentric delete-repo` command deletes the repository identified by the *<reponame>*. This command completely removes the repository -- including all its changes and its Git repository -- from the GitCentric installation. It also removes configuration information specified on the AccuRev Connector panel -- AccuRev server connection details, branch-stream mappings, and whether or not you are associating issues with commits.This command requires the **delete-project.jar** file that is installed to ***<gc_home>*/site/plugins**. GitCentric displays an error message if it cannot locate this file when you run the `delete-repo` command.

Cautions and Restrictions

Before executing the `delete-repo` command, be aware of the following cautions and restrictions:

- *You cannot undo a `delete-repo` command.* As such, you should use it only with extreme care, and ensure that you have taken normal precautions like backing up any important data.
- The `delete-repo` command removes the repository only on the server on which the command is run. In a master/slave environment, if you delete a repository from the GitCentric master server, you must also delete that repository from any of the slaves.
- You cannot delete a repository that uses a submodule subscription without first removing the submodule registration.
- You can run the `delete-repo` command only if you are a member of a group that has **Administrate Server** permissions.

Options

`<reponame>`
: The repository you want to delete.

`[--force]`
: Delete the repository even if it has open changes.

`--help, -h`
: Display the help for this command.

`[--noNotify]`
: Do not notify the GitCentric bridge of the configuration change.

`--yes-really-delete`
: Required in order to execute the command.

`--`
: "End-of-options" marker. See *Basic Syntax* on page 89 for more details.

Examples

Delete the "test_repo" repository regardless of whether or not it has open changes. Do not notify the GitCentric bridge.

```
> ssh -p 29418 mylogin@myserver gitcentric delete-repo test_repo --force
--noNotify --yes-really-delete
```

Delete the "acme" repository only if it has no open changes; notify the GitCentric bridge.

```
> ssh -p 29418 mylogin@myserver gitcentric delete-repo acme --yes-really-delete
```

Access

Only users in a group with **Administrate Server** permissions.

See Also

Basic Syntax on page 89, *config-repo*, *ls-repo*

ls-repo display GitCentric configuration

Usage

```
gitcentric ls-repo [--all] [--help] [<reponame>...] [--]
```

Description

The `gitcentric ls-repo` command lists GitCentric configuration information.

Specify one or more repositories to display with arguments *<repo_1>* through *<repo_n>*.

Note that only repos that are eligible to be mapped to AccuRev are listed, so a "parent-only" repo is not displayed, even if its children are. Use the `ssh gerrit ls-projects --type ALL` command to list all repos.

Options

`--all`
: Display all repositories that are accessible by the calling user. (Specifying `--all` is the same as specifying no options at all.)

`--help, -h`
: Display the help for this command.

`--`
: "End-of-options" marker. See *Basic Syntax* on page 89 for more details.

Examples

```
> ssh -p 29418 mylogin@myserver gitcentric ls-repo -all
```

Access

Any registered user. You will only see repositories to which you have at least Read access.

See Also

Basic Syntax on page 89, *config-branch*, *config-repo*

CLI Example

This example walks you through the entire process of creating a repository, cloning it, populating it with content, pushing, and finally mapping one of its branches to an AccuRev stream, all through Git, Gerrit, and GitCentric CLI commands.

1. Create a repository with an initial empty commit:

   ```
   ssh -p 29418 yourUsername@yourServer gerrit create-project --empty-commit repoName
   ```

2. Register the repo with GitCentric and associate it with an AccuRev server:

   ```
   ssh -p 29418 yourUsername@yourServer gitcentric config-repo repoName
       -S 10.0.0.182:5050 -u syncUser -p syncUserPassword
   ```

3. Clone the repo.

   ```
   git clone ssh://yourUsername@yourServer:29418/repoName repoName
   ```

4. Change to the new clone's directory and configure it:

   ```
   cd repo_tues2
   git config user.name "Firstname Lastname"
   git config user.email "f.lastname@yourdomain.com"
   git config remote.origin.push refs/heads/*:refs/for/*
   ```

5. Using your preferred text editor, create a file in the clone.

6. Add, commit, and push the file:

   ```
   git add --all
   git commit --all -m"First commit"
   git push
   ```

   Note: This step creates branch "master", which must exist before the next step.

7. Map the "master" branch to a stream named "agc_devel" on the AccuRev server:

   ```
   ssh -p 29418 yourUsername@yourServer gitcentric config-branch repoName
      -b master -s streamName -e /
   ```

8. Display a listing showing the mapped repo:

   ```
   ssh -p 29418 yourUsername@yourServer gitcentric ls-repo
   ```

D. GitCentric Glossary

A

access control list (ACL)

A GitCentric security feature consisting of a set of permissions that controls the access of groups to repositories. GitCentric group ACLs and AccuRev Element ACLs (EACLs) are two totally different mechanisms. See also *element access control list (EACL)*.

ASSIGN_USER_PRIVILEGE

A property of the AccuRev server configuration file, `acserver.cnf`, that identifies the groups and/or users allowed to perform AccuRev operations like ***promote***, ***chmod***, ***keep***, and ***move*** as another user using the ***-u*** command option (`accurev -promote -u <username>`, for example). In GitCentric, the *gcSyncUser* is typically identified using this property.

B

backed

In AccuRev, an element in a workspace or stream has “backed” status if it is not currently active there. This means that the workspace/stream inherits the version of the element that is currently in the workspace/stream’s parent stream (also called the backing stream).

branch

The active line of development in a Git repository. A Git branch is roughly equivalent to an AccuRev stream and is mapped to a stream in GitCentric.

bridge

The GitCentric server configured with the Tomcat web server that is part of the GitCentric installation package. The bridge is responsible for keeping an AccuRev stream synchronized with its mapped Git branch.

C

catalina.log

Apache Tomcat log in the **/tomcat/logs** directory where you installed the AccuRev Web User Interface used for troubleshooting a running GitCentric system.

change package

In AccuRev, a set of entries that records the changes to one or more elements that were made to implement the feature or address the bug described in an issue record. Each entry in the change package describes changes to one element: the changes between the basis version and the head version. A change package is sometimes referred to as "CPK".

checkout

In AccuRev, an operation that makes a file active in a workspace without recording any new changes to the file in the AccuRev repository.

CLI path

The full path to the AccuRev client executable that can communicate with the AccuRev server. For example: `/opt/accurev/bin/accurev`.

clone

In Git, the operation that creates a complete copy of a Git repository on your local directory.

commit

In Git, the operation that records changes in the local repository. See *pull* and *push*.

D

depot

The portion of the AccuRev *repository* that stores the entire history of a particular directory tree. The AccuRev depot is roughly analogous to a *Git repository*.

E

EID

See *element-ID*.

element-ID

The unique, immutable integer identifier by which AccuRev tracks the changes to a particular file or directory. An element's name or pathname can change, but its element-ID never changes.

element

A file or directory that is under AccuRev version control. See *version*.

element access control list (EACL)

An AccuRev security feature consisting of a set of entries ("permissions") that controls the rights AccuRev users (including the *service account*) have to AccuRev elements. Examples of EACL permissions include View Elements and Modify Elements. GitCentric access control lists (ACLs) and AccuRev EACLs are two totally different mechanisms. See also *access control list (ACL)*.

eventStream

An attribute of an AccuRev *stream* that controls whether or not a trigger fires when the stream content changes. Any stream mapped to a Git branch must be an event stream.

eventStreamHWM

The transaction number of the last change made to the content of an AccuRev *stream*. The last change made to a stream is referred to as the stream's *high water mark*, or HWM.

export

The GitCentric process used to populate a Git branch with the contents of the mapped AccuRev stream. An *initial export* occurs during repository creation using the GitCentric GUI. *Incremental exports* occur each time you promote changes to a mapped stream.

See also *import*.

F

file storage area

The portion of an AccuRev depot in which AccuRev maintains a permanent copy ("storage file") of each newly created file version. See *metadata*.

G

gcSyncUser

The name given to a special service account user, used for communication between GitCentric and AccuRev. The gcSyncUser is typically a member of an AccuRev group such as `scm_bridge_group`, which is assigned to the *ASSIGN_USER_PRIVILEGE* property in acserver.conf.

Gerrit Code Review

A free, web-based team software code review tool developed at Google that integrates with Git version control software. It is incorporated into AccuRev's GitCentric product.

Git

A distributed revision control and source code management (SCM) system initially designed and developed by Linus Torvalds.

Git repository

A distributed data structure that contains a local copy of the central repository with complete history and tracking facilities.

See also *clone*.

gitcentric.war

The .war file for the GitCentric GUI, which includes the open-source, third-party Gerrit Code Review package. Expanded in the `tomcat/webapps` directory where you installed GitCentric.

GitCentric database

A database on the AccuRev server that contains the AccuRev stream-Git branch mappings as well as a history of all import and export activity.

GitCentric stream

An AccuRev stream that is mapped to a branch in a Git repository (and is an event stream).

H

high water mark

The last change made to a stream is referred to as the stream's *high water mark*, or HWM. See *eventStreamHWM*.

hook

In Git, a way to fire a custom script to perform some action. A Git hook is analogous to an AccuRev trigger.

I

import

The GitCentric process used to populate or update an AccuRev stream with the contents of the mapped Git branch. An *initial import* occurs during stream creation using the GitCentric GUI. *Incremental imports* occur each time you push commits to the central repository.

See also *export*.

inherit

In AccuRev, the facility by which versions in higher-level streams automatically propagate to lower-level streams. If an element is not currently active in a stream or workspace, the stream or workspace inherits the version of the element that appears in its parent stream.

K

kando

The engine that powers GitCentric. It provides a framework for synchronizing data between the AccuRev SCM and other products such as Git.

kandoMaintain

A command line utility used to perform occasional administrative operations

M

map ("mapping")

An association between an AccuRev stream and a Git branch. See also *mount point*.

merge

In AccuRev, an operation that combines the contents of two versions of the same element. To merge the contents of text files, AccuRev uses a "3-way merge" algorithm: it compares the two files line-by-line with a third file -- the version that is the closest common ancestor of the other two.

metadata

Information stored in the AccuRev repository other than the contents of file versions. Metadata is stored in the repository database; file contents are stored in the *file storage area*.

mount point

The directory within an AccuRev stream that is mapped to a Git branch. Access to directories and files in the stream are restricted to the contents of the mount point and its child directories.

O

overflow

A condition identified during a restore of the GitCentric database. An overflow condition occurs when the GitCentric database is backed up before the Git repositories are copied. In this event, it is possible that the GitCentric database and the Git repository copies are out of synch -- specifically, the GitCentric database probably does not have a complete set of AccuRev transaction-Git commit pairs reflecting the latest commits in the Git repositories.

P

parent stream

In AccuRev, the stream that is just above a given workspace or stream in a depot's stream hierarchy. The given workspace/stream inherits versions from the parent stream. (Also "backing stream" and "basis stream".)

project

In Gerrit Code Review, "project" often refers to a repository, such as "Watched Projects". In AccuRev GitCentric, project is used in its more generic sense: a planned or defined undertaking.

promote

In AccuRev, the operation that transitions a version from being active in one workspace or stream to being active in the parent stream (or some other stream). The promote operation creates a new *virtual version* in the parent stream. The virtual version provides an alias for the real version -- the version that was originally created in some user's workspace. The promote operation is analogous to the push operation in Git.

pull

In Git, the process of refreshing your Git clone by copying the latest commits from the central Git repository. The pull operation is analogous to the update operation in AccuRev.

Tip: Consider using **git pull --rebase** to provide serialized commits, avoiding the need to separately merge remote changes in your branch.

push

In Git, the process of updating the central Git repository by copying commits from your Git clone to the central Git repository. The push operation is analogous to the promote operation in AccuRev.

R

repository

See *clone* and *Git repository*.

rollback

A condition identified during a restore of the GitCentric database. A rollback condition can occur when there is a large interval of time between when the copy of the Git repositories was created and the GitCentric database backup was performed. In this event, the GitCentric database will have mapped Git commits of which the repositories are unaware.

S

service account

The AccuRev user account that is used by a specific branch-stream GitCentric mapping to synchronize Git repositories and AccuRev streams. By convention, the service account is an AccuRev user created for this role named "`gcSyncUser`" and is a member of a group such as "`scm_bridge_group`", which is set to the `ASSIGN_USER_PRIVILEGE` property in the AccuRev `acserver.conf` file. Each branch-stream mapping can have its own service account with different permissions to control access to data. See *gcSyncUser*.

SHA-1

The Secure Hash Algorithm (SHA) in Git that refers to the unique identifier, or name, given to each object in the Git object model -- blobs, trees, commits, and tags.

snapshot

In AccuRev, an immutable stream that captures the set of element versions of another stream at a particular time. A snapshot cannot be renamed or modified in any way. This term can also be used to describe an import of a repository without commit history into an AccuRev stream.

SSH

The Secure SHell (SSH) network protocol for secure data communication between two networked computers. In GitCentric, SSH is used for communication between local Git repositories and the central Git repository using public/private key pairs.

stream

In AccuRev, the data structure that implements the set of element versions in a particular depot. A stream is roughly equivalent to a Git branch and is mapped to a Git branch in GitCentric.

T

transaction

A record in the AccuRev repository that indicates a particular change: promoting of a set of versions, changing the name of a stream, modifying an issue record, and so on. Each transaction has an integer transaction number, which is unique within the depot.

U

update

In AccuRev, the operation that copies new versions of elements into a workspace from its parent stream. The update operation is analogous to the pull operation in Git.

V

version

In AccuRev, a particular revision of an element, reflecting a content change (files only) or a namespace change (files and directories). All versions are originally created in workspaces, and can subsequently be promoted to dynamic streams. The original (workspace) version is termed a *real version*. Each promotion to a dynamic stream creates a *virtual version*, which serves as an alias for (pointer to) the original real version.

W

workspace

In AccuRev, a location in which one or more users performs his or her work on files that are under AccuRev version control. Each workspace consists of a workspace stream in the AccuRev repository and a workspace tree in the user's disk storage.

Index

A

access rights (ACLs)
  AccuRev element ACLs (EACLs) 51
  group-based ACLs 34
AccuRev
  configure 28
  depots and case-sensitivity 31
  using Git reserved names in 39
AccuRev server
  adding to GitCentric 46
ACLs 4, 10, 11
  EACLs (AccuRev Element ACLs) 8, 10
  GitCentric group-based ACLs 10, 29, 30, 31, 34, 38
addconfig, kandoMaintain 77
Administration page 65
ASSIGN_USER_PRIVILEGE 4, 39

B

backing up GitCentric
  best practices 83
  commands for 83
  Git repositories and 84
  GitCentric databases and 84
  how to 84
  what gets backed up 84
backup and restore
  commands for 83
backupdbs (kandoMaintain) 76
branch
  create 34
  remove 31
branches
  choosing a mount point when mapping to a stream 40
  comparing commits across branches 25
  viewing commit history 21

C

change packages (AccuRev)
  enabling 43
  error messages 45
  using git pull --rebase 46
checksync (kandoMaintain) 77
CLI commands
  basic syntax 89
  config-branch 90
  config-repo 92
  delete-repo 94
  ls-repo 96
  spaces and quoting 89
CLI path 47, 74
clones
  configure for direct push 20
  configuring for Code Review 20
  creating 19
  troubleshooting 20
Code Review 61
command syntax 89
commit-msg hook
  enabling Gerrit Code Review and 52
commits
  reviewing changed files in a commit 23
  view commit history 21
Commits page
  gravatars and 22
config-branch CLI command 90
config-repo CLI command 92
configure
  AccuRev 28
  AccuRev server 47
  GitCentric 29
contacting technical support viii
cronjobs
  setting up garbage collection 55
crontab
  setting up cron jobs for garbage collection 55

D

databases
  backing up 84
  GitCentric databases 84
delete-repo CLI command 94

E

email addresses

entering contact information 59
preferred 59
export
troubleshooting 54

F

file viewer
about 23
files
reviewing changed files in a commit 23
full_backup.sh
backing up GitCentric with 84
full_restore.sh
restoring GitCentric with 85

G

garbage collection
benefits 55
setting up 55
Gerrit Code Review 1, 11, 26, 34, 37, 49, 58, 61
allowing self-reviews 52
commit-msg hook and 52
disabling 53
enabling 52
enabling replication 53
setting up garbage collection 55
support for Gerrit hooks 73
Git
support for Git hooks 73
using reserved names in AccuRev 39
git commands
using git pull --rebase 46
GitCentric
backing up 84
build SHA displayed on Code Review page 62
overview 1
restoring 83
gravatars
Commits page and 22
Source page and 25
used in GitCentric 22, 25
groups
administration 73
managing 49

H

help, kandoMaintain 77
hooks
support for hooks in GitCentric 73
how to
Add an AccuRev server 46
compare branches 25
Configure Access Rights (ACLs) for a Repo 34
Create a Clone from a GitCentric Repository 19
Create a Repository for GitCentric 29
Create an SSH key 14
Create Branches for a Repo 34
determine how divergent a branch's commits are 25
disable Gerrit Code Review 53
Enable and Use Change Packages 43
enable Gerrit Code Review 52
Import a Snapshot 32
Import an Existing Git Repo 31
Log into GitCentric 14
Manage GitCentric Groups 49
Map a Git Branch to an AccuRev Stream 38
Register with GitCentric 16
Remove a Branch or a Repository 31
review a commit's changed files 23
Set General Attributes for a Repo 33
Set Preferences 18
Unmap a Git Branch from an AccuRev Stream 42
view commit history 21

I

import 31
snapshot 32
troubleshooting 54

K

kandoMaintain 75
addconfig 77
backupdbs 76
checksync 77
help 77
lsconfig 77

mkdb 75
mvrepos 76
restoredbs 76
rmconfig 77
rmdb 76
testconn 77
upgradedb 76

L

logging in to GitCentric 14
lsconfig (kandoMaintain) 77
ls-repo CLI command 96

M

mapping
behavior 5
Git branch to AccuRev stream 38
keeping Git and Accurev in sync 5
rules 4
unmapping 42
mapping branches to streams
choosing a mount point 40
memory
using garbage collection to reduce waste 55
mkdb (kandoMaintain) 75
mount point 28
considerations for choosing 40
mvrepos (kandoMaintain) 76

O

overflow condition
during restore operation 87

P

performance
using garbage collection to improve 55
Port (AccuRev) 15, 47
preferences 18
preferred email addresses 59
procedures
administrators only 28
all users 13
pull
using git pull --rebase 46

R

regular expressions 45
for commit comments 43, 45, 92
replication
enabling Gerrit Code Review replication 53
repositories 66
backing up 84
deleting using the command line 94
removing 31
repository
create 29
remove 31
set general attributes 33
reserved names
Git reserved names and AccuRev 39
restoredbs (kandoMaintain) 76
restoring GitCentric
backup and restore 83
overflow condition 87
rollback condition 87
usage scenarios 87
rmconfig (kandoMaintain) 77
rmdb (kandoMaintain) 76
rollback condition
during restore operation 87

S

security
configuring 10
server
adding 46
Server Name (AccuRev) 47
Service Account 8, 39, 51
Service Sccount 104
site header (Code Review) 18
Source Tree page
gravatars and 25
SSH
CLI commands 89
creating a key 14
Status Monitor
description 41, 69
displaying 41, 69
reference information 69
streams

choosing a mount point when mapping to a branch 40
support
contacting technical support viii
synchronizations
Status Monitor for 41

T

technical support
contacting viii
testconn (kandoMaintain) 77
troubleshooting
clone issues 20
Code Review issues 63
GitCentric restore operation 87
import/export 42, 54
restore operation 87
typographical conventions viii

U

upgradedb (kandoMaintain) 76
username, for GitCentric automation 39

V

version
build SHA displayed in the GitCentric GUI 62