**Credentials in Jenkins**

In addition to globally securing different aspects of Jenkins, using specific, secure credentials forms a key part of having a secure Jenkins environment. The Credentials plugin (included with installations of Jenkins) providesmechanisms for users to create and manage credentials, as well as an API for plugins to use to store and access credentials.

It’s worth saying a word here about what we mean by the general term “credentials.” Often you will hear this also described as a “secret.” In general, we mean any value or values that provide access to a restricted resource. A list of the credential types includes:

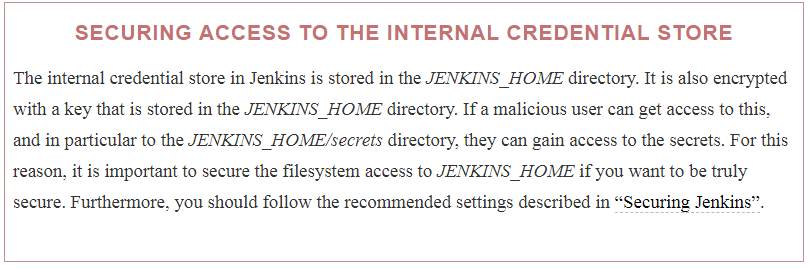
* Usernames with passwords—may be conjoined when used (treated as one item) or separated
* Docker certificates directories (now deprecated)
* Docker host certificate authentications
* SSH usernames with private keys
* Secret ZIP files—ZIP files with the credentials
* Secret files—flat files with the credentials
* Secret texts—tokens or other chains
* Certificates—Java KeyStores with the certificates/certificate chain

Specific examples might include:

* A username and password combination to gain access to a source control repository
* A digital key and certificate to sign an entity
* A secret text string that can be matched to identify that content is from a specific source
* An SSH key set to deploy to a server

Other types of credentials could include less formalized items, such as binary data, or more formalized ones, such as OAuth credentials.

Once created, credentials have to be stored somewhere. The Credentials API allows for accessing an external credential store (an application capable of storing and retrieving credentials). However, Jenkins has an internal encrypted credential store that is used by default.



One other fundamental point about credentials is that they are associated with a *context*. Contexts represent a way of thinking about the different entities that make up Jenkins as a hierarchy. The root context is Jenkins itself. Other contexts include jobs, users, build agents, and folders. Additionally, plugins can define new contexts.

With this background, we can delve more into the characteristics and properties associated with managing credentials in Jenkins. The first one we’ll look at is the credential’s *scope*.

## **Credential Scopes**

Credentials have a scope associated with them. This is a way to say how they can be exposed. There are three main scopes that Jenkins uses:

System

As the name implies, this scope is associated with the root context, the Jenkins system. Credentials in thisscope are only exposed to system and background tasks and may be used to do things such as connect to buildnodes/agents.

Global

The global scope is the default scope and the one to use generally to ensure that credentials are available tojobs in Jenkins. Credentials in this scope are exposed to their context and all child contexts of that context. (Recall that credentials are associated with a context and that contexts represent a hierarchical structure of the main parts of Jenkins.)

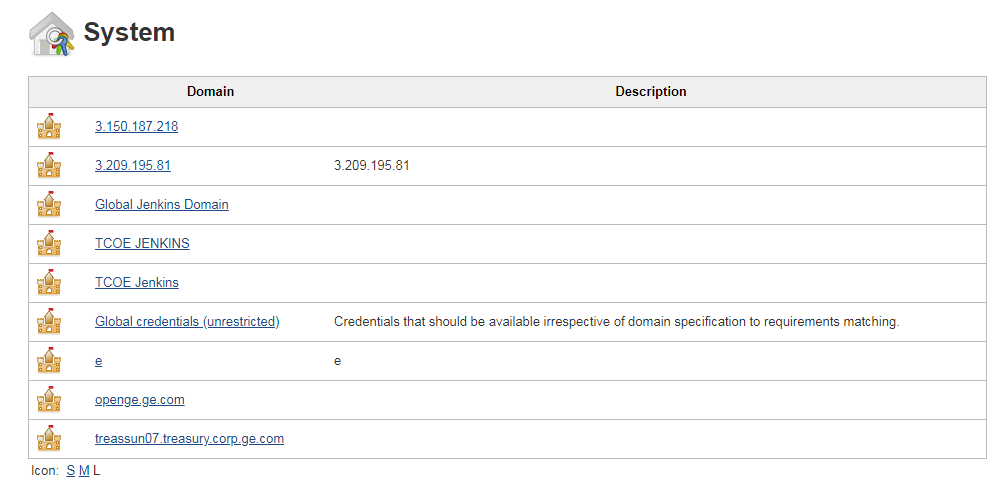
User

As the name implies, this scope is per-user. This means that the credentials are only available when threads in Jenkins are authenticating as that user.

## **Credential Domains**

Credential domains provide a way to group together, under a common domain name, sets of credentials. Typically, the common domain name will imply some functionality or application type that the credentials are expected to work with.





When you define a credential domain, you provide a domain name and a “specification” such as a hostname or URL pattern.



Jenkins always has at least one credential domain—the global domain. The global credential domain has no specification, so it is available for anything in Jenkins to use.

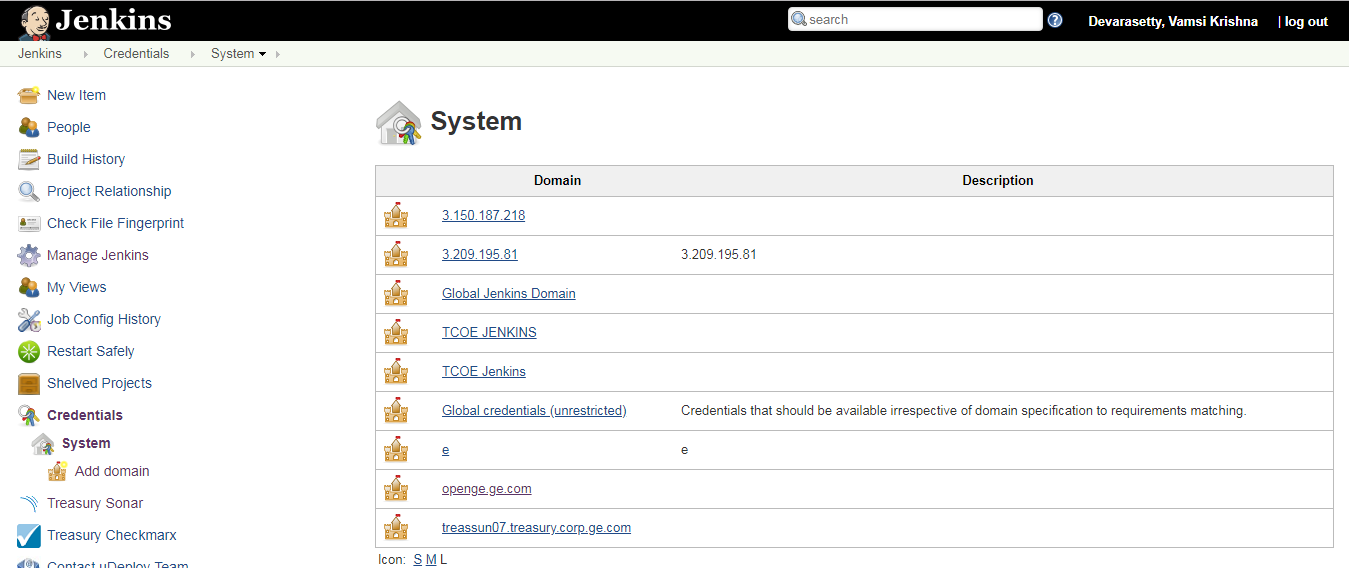
**Credential Providers**

A credential provider is a place where credentials can be stored and retrieved. This can be an internal credential store or an external credential vault.

There are several standard credential providers. These are:

System credentials provider (Jenkins credentials provider)

This exposes credentials at the root context (Jenkins itself). Two credential scopes are available: system and global. To look at this, you can go to Jenkins → Credentials → System.



User credentials provider

This exposes a per-user credential store for a user. Only the user scope is available, and a user cannot see theper-user credentials of another user. To see these credentials, you can either go to Jenkins → <*username*> → Credentials → User or Jenkins → People → <*username*> → Credentials → User.

Folder credentials provider

This is provided by the Folders plugin. It exposes a per-folder credential store and supports the global scope for the folder and any children. To see these credentials, go to Jenkins → <*folder name*> → Credentials → Folder.

BlueOcean credentials provider

This scopes credentials to the Blue Ocean interface and items created/accessed directly through it.

## Credential Stores

Credential stores allow credential providers to expose credentials to Jenkins. Stores are associated with a specificcontext and are either tied to the global domain or can use a custom domain. They can support a set of credential domains.

Internal stores store the actual credentials. External stores will typically be either a simple flat reference of credentials or a service with metadata and more advanced features like querying.

# Administering Credentials

Administration for credentials can be done through the Configure Credentials interface, accessible under the Manage Jenkins menu. The options on this screen allow a Jenkins user to:

* Select which credential providers will be available to Jenkins to resolve credentials.
* Select the types of credentials that can be resolved and configured.
* Specify the types of credentials that can be included or excluded for a specific provider.

# 

## Selecting Credential Providers

# At the top of the Configure Credentials screen is a drop-down list to tell Jenkins which credential providers it can use. The default choice is to use “All available” providers. However, if you need to subset the list by including or excluding certain providers, there are options to do that.

# 

# If either the “Exclude selected” (exclude providers) or “Only selected” (include providers) option is chosen, a list of providers with checkboxes is displayed. Depending on the option, the checkboxes next to the appropriate providers can be checked to either exclude them from the set of available providers or include them in the set of available providers

# 

## Selecting Credential Types

# Just as the subset of credential providers can be chosen, the next section on the screen allows you to select the set of credential types that Jenkins can use. The default choice is to use “All available” types. However, if you need to subset the list by including or excluding certain types, there are options to do that.

# 

# If either the “Exclude selected” (exclude types) or “Only selected” (include types) option is chosen, a list of types is provided with checkboxes. Depending on the option, the checkboxes next to the appropriate types can be checked to either exclude them from the set of available types (see [Figure 5-12](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_selc_spectype_credex)) or include them in the set of available types.

# 

## Specifying Credential Types by Provider

# The last part of the Configure Credentials screen is the Restrictions section. This allows you to specify the types of credentials Jenkins will allow or exclude from a specific provider (see [Figure 5-13](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_refres_credtyp_credprov)). This is a way to fine-tune what Jenkins can use from a provider. Note that doing this is optional and not required.

# 

# The Add button in this section has two options (“include” and “exclude”). Selecting either will create a new page element allowing you to select a provider and then a type. If you have selected “include,” this type of credential will be included for that provider, and vice versa for “exclude.”

# If you need to set up restrictions for multiple types and/or multiple providers, adding all of the elements can take some time. However, as noted earlier, using this is optional, not required.

# 

# Creating and Managing Credentials

# Earlier, we discussed the notion of “contexts” in Jenkins. Each context in Jenkins that has an associatedcredential store will also have a credential “operation” added to it by the Credentials plugin. That means that, by default, you will have Credentials menu items specific to the system, user, and folder contexts.

# Management for system-level credentials can be accessed simply by selecting Credentials from the top level of Jenkins

# 

# Remaining read from the book

# Advanced Credentials: Role-Based Access

While the common credentials options will handle many use cases, there may be times where you want to use amore granular approach to security and authorization. An example use case would be creating new roles with aset of specific permissions assigning roles to individual users. The [Role-based Authorization Strategy plugin](https://plugins.jenkins.io/role-strategy) is designed to provide this kind of functionality.

More specifically, the plugin allows for the definition of three types of roles:

Global roles

Roles that span across projects with permissions such as Job, Run, and SCM

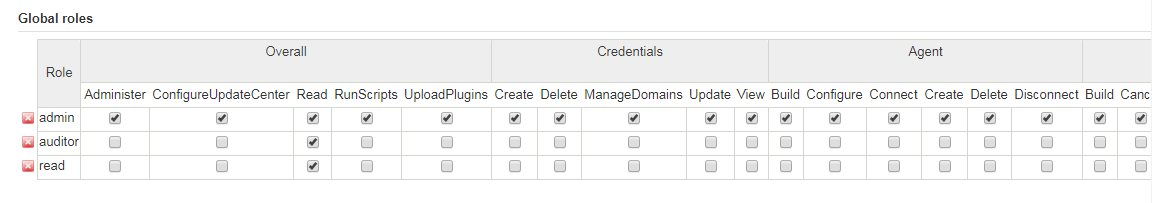
Project roles

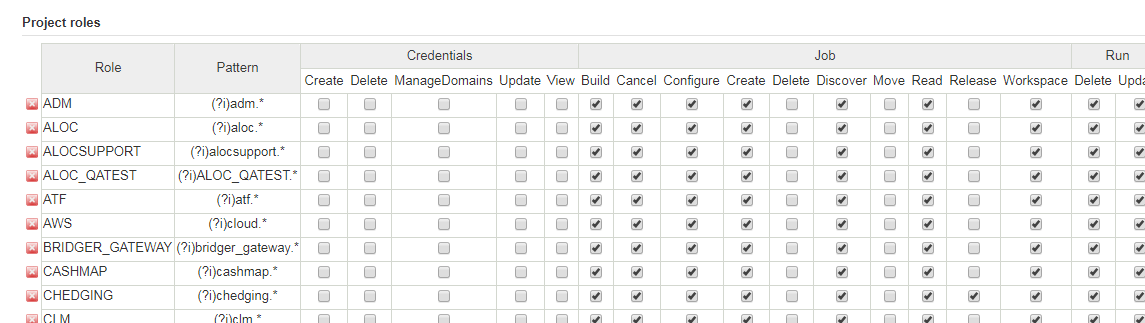
Roles particular to a project from the Job or Run category

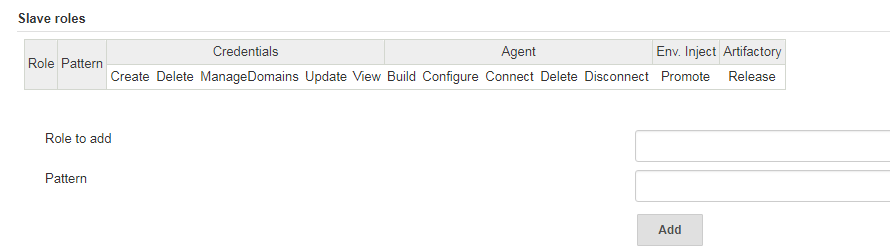
Slave roles

Roles with permissions to administer nodes

The plugin also provides a macro facility so that macros can be used as criteria for what roles apply to.

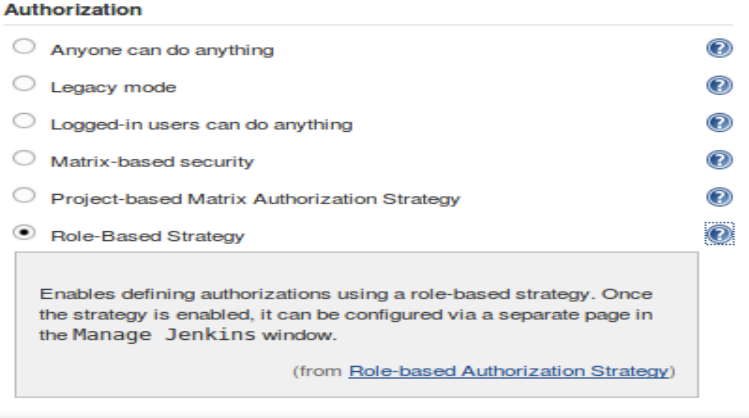




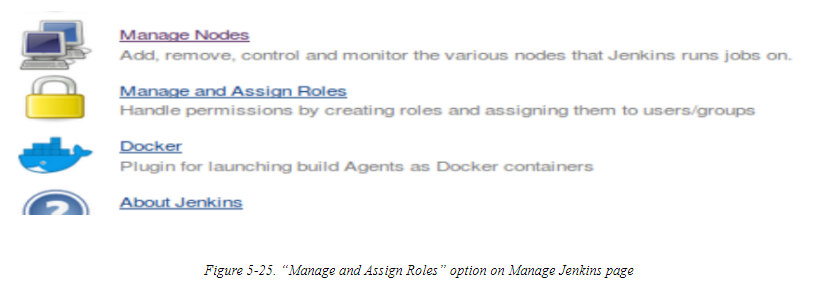


## **Basic Use**

Installation of the plugin is the same as for any other Jenkins plugin. Once installed, if security is enabled in Jenkins, there will be a new option named Role-Based Strategy under Authorization in the Access Control section of the global security configuration page



If this option is selected and saved, there will then be a new selection on the Manage Jenkins page named “Manage and Assign Roles” ([Figure 5-25](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_entry_2manage)). This is the gateway to the plugin’s functionality.



On the Manage and Assign Roles screen are three selections for the main functions: Manage Roles, Assign Roles, and Role Strategy Macros ([Figure 5-26](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_manage_assignroles)). We’ll look at each of those in more detail in the next sections.

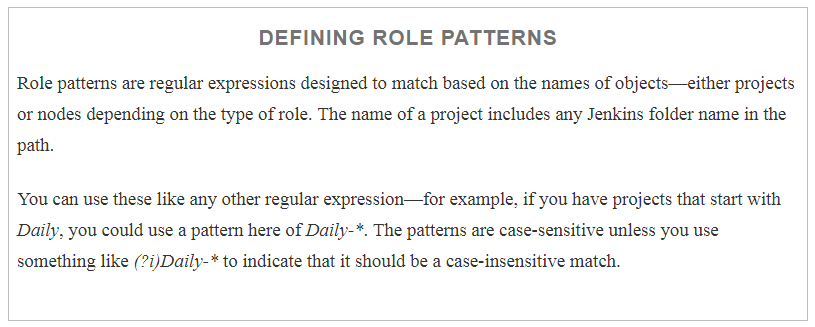


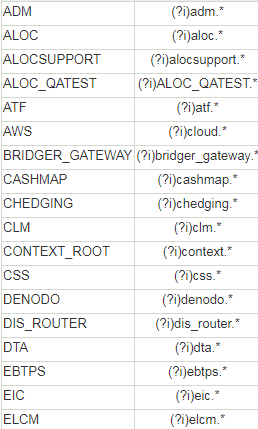
## **Manage Roles**

As the name implies, this screen allows you to create or delete roles and assign permissions to them. There are three sections here for each of the three kinds of roles mentioned earlier: global, project, and slave.

The mechanics of using each section are similar to the Jenkins matrix–based authorization model. There is a matrix where each row contains a defined role and each column is a specific permission within a category of Jenkins object (Overall, Credentials, Agent, etc.). To grant a permission to a role, you simply click on the checkbox for the column of the desired permission in the row for the role. If a checkbox is blank, that indicates the role does not have that permission. To remove an existing permission for a role, simply uncheck the box in the appropriate column.

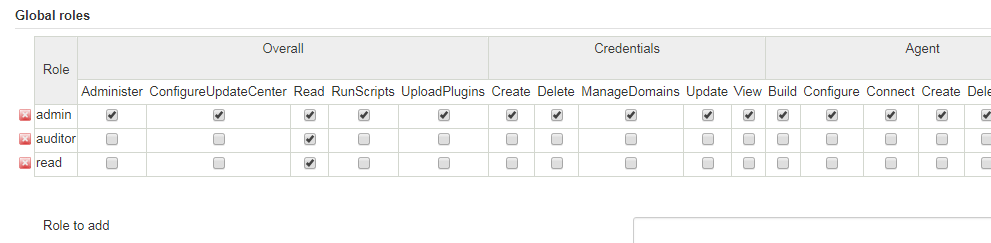
You can create a new role by entering the desired role name in the “Role to add” box. The Project and Slave sections also expect a pattern. These patterns are used to associate the project or slave role to matching project names or node names, respectively. The Global section does not require a pattern, since we assign specific users to those roles rather than relying on matching user IDs. The following note details more about the syntax of the patterns.



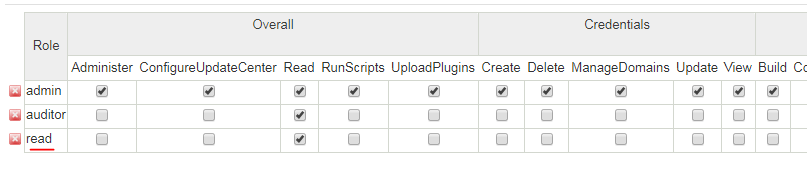


### GLOBAL ROLE EXAMPLE

By default, we have an *admin* role that has all permissions. To add a new role, we simply type in the desired role name in the “Role to add” box and click Add. In this case, let’s suppose we want to create a new *job-admin* role. The idea is that this role can administer things around jobs. It does not need (and should not have) all the permissions of the traditional admin role.



After adding the role, we can check the appropriate boxes to give the role the desired permissions, as shown in



PROJECT EXAMPLE

Carrying our example further, let’s suppose that we have two main types of jobs that we run on our Jenkins instance—daily and weekly. We want to define a role of *daily-job-admin* to allow a subset of people to administer the daily jobs but not the weekly jobs. Our daily jobs all have names or folder paths that start with *daily*, so we can use that for a pattern. [Figure 5-30](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_def_new_dai_projrole) shows the initial steps to set this up.

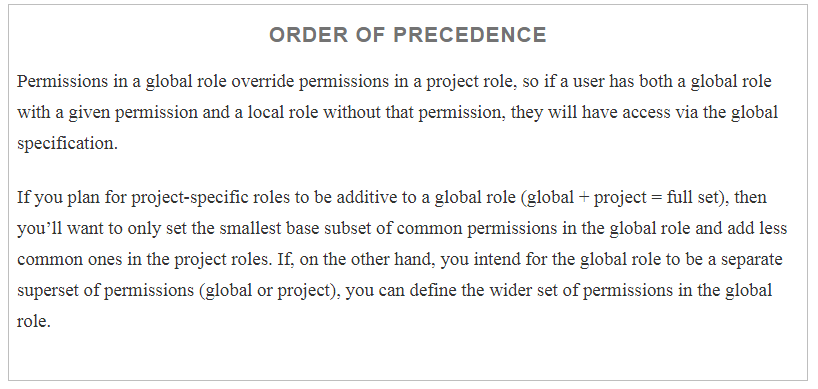


Once we add the new project role based on the project pattern, we can select permissions for the role just as we did for the global one ([Figure 5-31](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_asgn_perms_projrole)). However, because we’ve supplied the pattern, users with this role will only have the selected permissions for jobs matching that pattern.



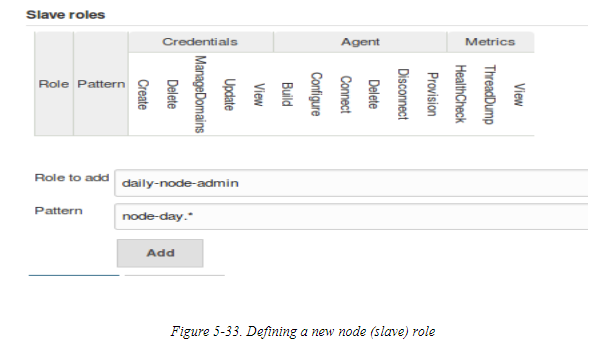
To round out our example, we would also add a *weekly-job-admins* role and roles for the daily and weekly users (non admins). An example of the completed list is shown in [Figure 5-32](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_wkly_roles_add4proj).



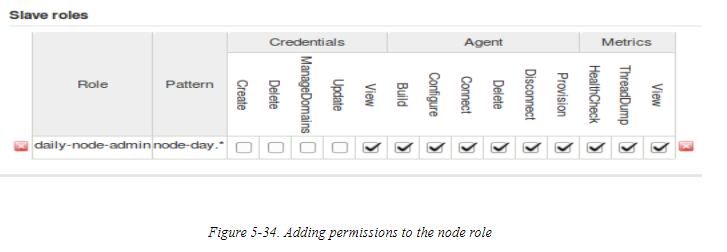


### SLAVE ROLE EXAMPLE

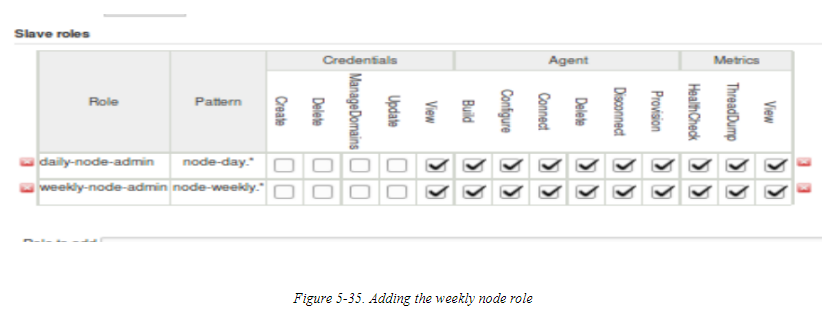
In addition to defining global and per-project roles, we can also define roles around the administration of nodes. This is done via the last section on this screen. [Figure 5-33](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_def_new_node_role) shows an example of adding a new role to administer nodes with names starting with *node-day*. (This pattern identifies nodes that we are using to run our daily jobs.)



Once added, we can assign permissions to the role just as with the previous sections for global and project roles

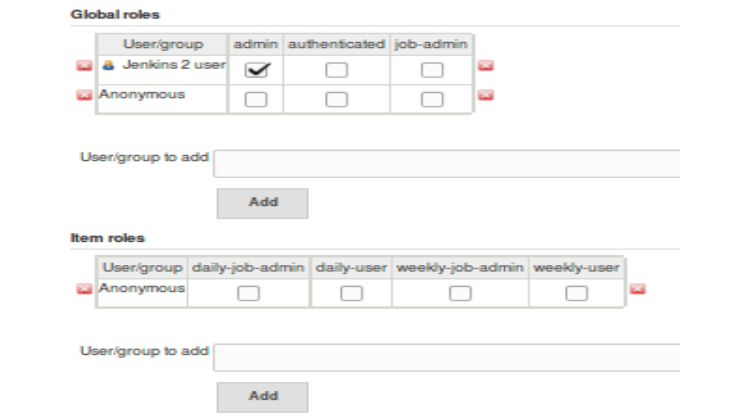


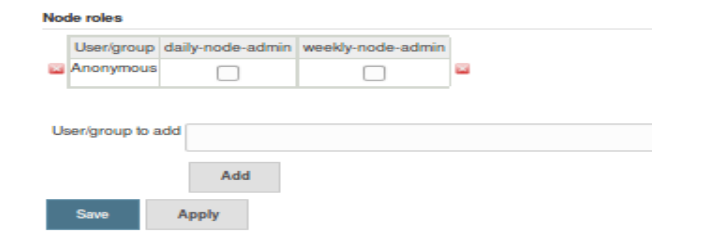
To finish out our example model, we can add a project role for the jobs that run weekly, and a slave role for users who can administer the nodes that run the weekly jobs



## Assign Roles

Once we have our desired roles set up, we can assign users or groups to particular roles. We do this using the Assign Roles screen, accessible from the “Manage and Assign Roles” page). For each category of role on the Manage Roles page, we have a corresponding section on the Assign Roles page. However, corresponding sections on the latter have the more “modern” names—“Item roles” and “Node roles.” To be clear, “Item roles” here corresponds to “Project roles” and “Node roles” corresponds to “Slave roles.” [Figure 5-36](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_asgn_roles_scrn) shows an example of a starting page for assigning roles.

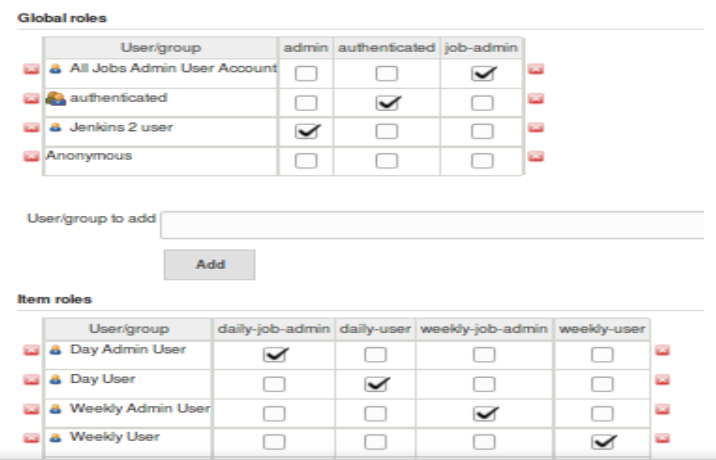


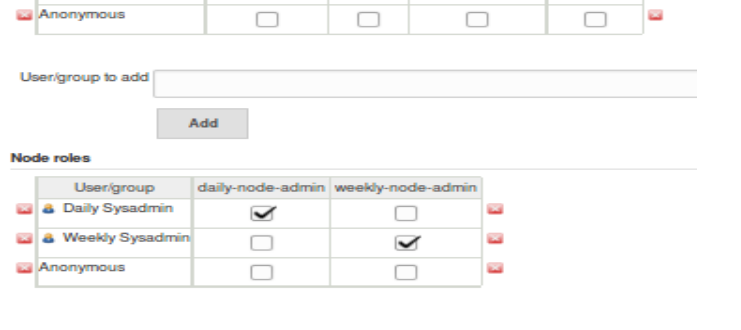


Usage here is straightforward. Within each section (Global, Item, and Node), the rows represent users or groups, and the columns represent the roles that have been defined for that category. Note that there is a default entry for the *Anonymous* user. Any other users/groups already defined will have rows as well.

To allow a user/group to have the permissions associated with a role, you simply enter the user/group name into the “User/group to add” text box, click the Add button, and then check the boxes in the columns corresponding to the roles you want them to have.

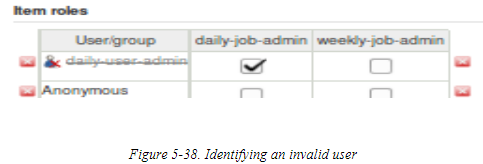
For example, suppose that we have the following user IDs: *all-jobs-admin*, *day-admin-user*, *day-user*, *weekly-admin-user*, *weekly-user*, *sysadmin-daily*, and *sysadmin-weekly*. The “admin” user IDs are intended to be the administrators for their respective categories. Once we fill in the particular users to match up to the intended categories, we will have a configuration like the one in [Figure 5-37](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_fin_conf_asgnroles).





### DEALING WITH INVALID USERS

The forms for assigning users will allow you to type in and add any user/group name initially. Once you save your changes, validation will be done to make sure the user/group is valid. If it is not, then when you go back into the Assign Roles page, you will see the user/group name with a line through it—indicating the user/group doesn’t exist or isn’t valid ([Figure 5-38](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_id_invalids)). At that point, you can delete the user/group by clicking on one of the small, red “X” symbols on either end of the row.



## Role Strategy Macros

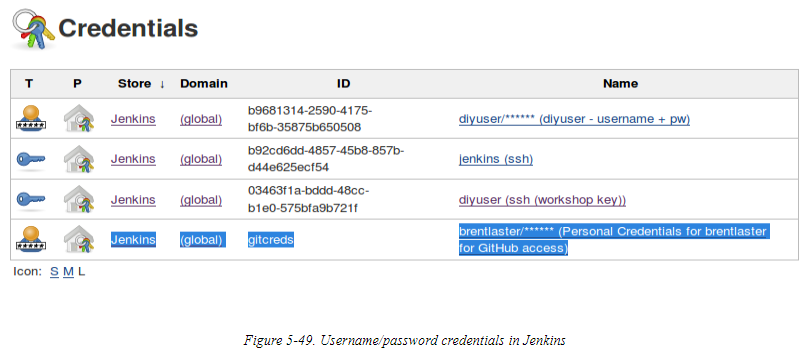
Read the book

# Working with Credentials in the Pipeline

There will be times you’ll need to supply credentials in your pipeline for your pipeline steps. In this section, we explore some Pipeline constructs for working with the basic types of credentials.

## Username and Password

First, we want to make sure we have the [Credentials Binding plugin](https://plugins.jenkins.io/credentials-binding) installed. Then we’ll define a set of credentials with a username and password in Jenkins



We can now use the withCredentials block in our pipeline to work with the designated credentials. Thesyntax for this block starts with the following:

withCredentials([usernamePassword(credentialsId: '<ID>',

passwordVariable: '<variable to hold password>',

usernameVariable: '<variable to hold username>')])

The idea here is that whatever variables are used for usernameVariable and passwordVariable will be filled in the username and password from the credentials specified by credentialsId.

## SSH Keys

To use SSH credentials in our pipeline, we can use the withCredentials block again, as shown here:

withCredentials([sshUserPrivateKey(credentialsId: '<credentials-id>',

keyFileVariable: 'MYKEYFILE',

passphraseVariable: 'PASSPHRASE',

usernameVariable: 'USERNAME')])

{

*// some block*

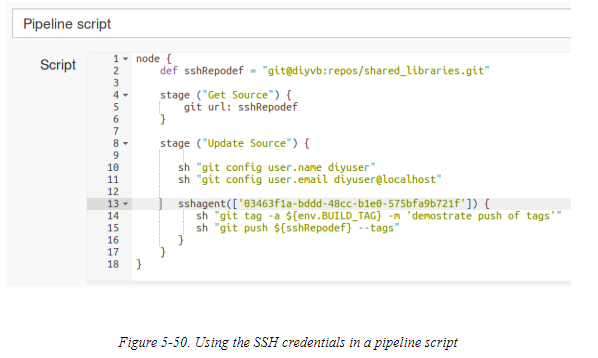
}

As an alternative, we can use an sshagent block. For this, we first need to make sure we have the [SSH Agent plugin](https://plugins.jenkins.io/ssh-agent) installed.

Now, we can use the sshagent block to do our access, passing in the credentials ID:

shagent([<credentials id>]) { }

Below shows an example of using this in a pipeline script.



## Token Credentials

When working with other types of credentials, the same general idea (using the withCredentials block) applies. The following is an example of using a token credential modeled on an example in the Jenkins documentation:

node {

withCredentials([string(credentialsId: '<token>', variable: 'TOKEN')])

{

sh '''

set +x

curl -H "Token: $TOKEN" https://some.api/

'''

}

}

A couple of points are worth mentioning about this:

* The shell script uses the triple quotes to handle a multiline script inline. (You can discover more about usingthe *sh* step in [Chapter 11](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch11.html#CH_Integration_with_the_OS).)
* The set +x prevents echoing out the credential as the script executes.

For other types of credentials, you can use the Snippet Generator for the withCredentials step and fill in thedesired binding.

# Controlling Script Security

# The pipeline functionality introduces the ability to run any arbitrary script. With this increased flexibility toexecute commands and do processing comes an increased importance of being able to control script security. In Jenkins 2, this security is provided by the [Script Security plugin](https://plugins.jenkins.io/script-security).

# 

# By default, users with the Overall/Administer permission can write or run whatever scripts they want. This level of permissions is equivalent to admin permissions on the Jenkins instance, and so is not appropriate for all users. So, Jenkins 2 includes two mechanisms to help with script security: script approval and Groovy sandboxing.

# DEPRECATED PERMISSIONS

# In previous versions of the role-based access/matrix plugin, there were additional permissions that could be set:

* Overall/Run Scripts
* Overall/Upload Plugins
* Overall/Configure Update Center

# This was deemed a security risk because these permissions were as powerful in some cases as the Overall/Administer permissions, so now you need to have the Overall/Administer permission to automatically be able to run scripts without approval.

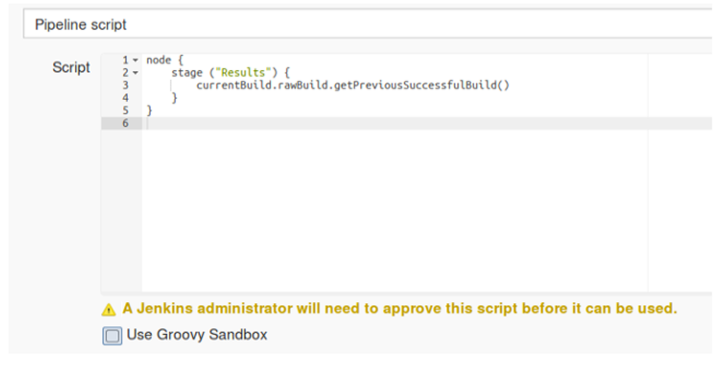
# If you do need to go back to the old insecure permissions for some reason, the org.jenkinsci.plugins.rolestrategy.permissions.DangerousPermissionHandlingMode.enableDangerousPermissions system property can be set to true.

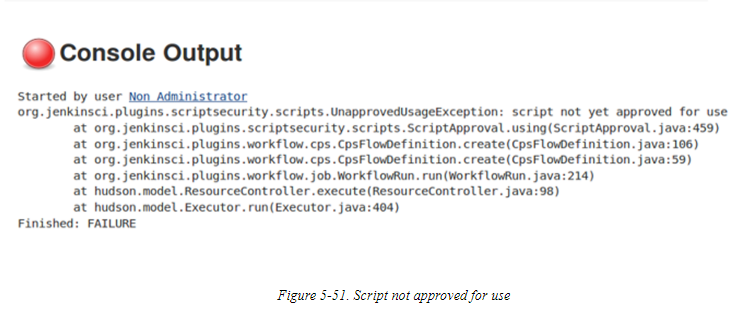
## Script Checking

When a Jenkins administrator creates a script or includes a script in a configuration and saves it, the script is automatically approved and added to an approved list. Those scripts in the approved list can be run by anyone. If a non administrator tries to run a script and it is not one in the approved list, then it is prohibited from running until/unless approved by an administrator.

The reason for this is that, unlike filling in web forms, scripts can (attempt to) do any arbritrary operations, including referencing internal objects in Jenkins. This could be a security risk as well as a technical risk, depending on what the script is trying to do.

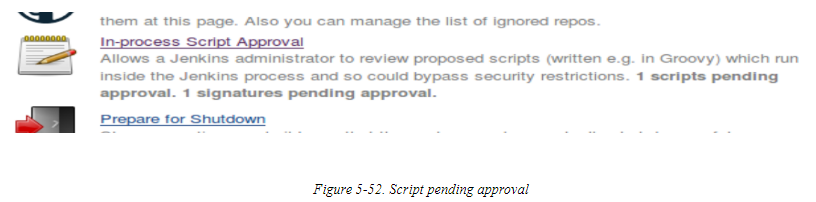
An example of a script that needs to be approved is shown in [Figure 5-51](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_scrip_not_4use). This one is flagged because it is trying to use the internal rawBuild object to get information. The figure also shows the output from trying to run the script—note the error message.



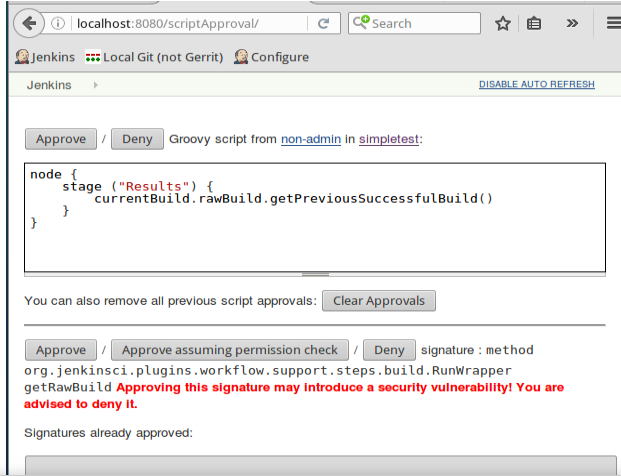


## Script Approval

If a nonadministrator tries to run a script that needs approval, Jenkins will prohibit running it. It will also add a notice about the need for approval to a queue, for an administrator to review. An administrator can then log in to Jenkins and go to Manage Jenkins → “In-process Script Approval.” An alert of the form “1 scripts pending approval” will be waiting for the administrator ([Figure 5-52](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_scrip_pend_approv)).



Once the administrator goes to the script approval area, they will have an option to approve or deny executing the script. The upper part of [Figure 5-53](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_acrip_approv_interadmin) shows this part of the form



# Groovy Sandboxing

# While the script approval mechanism provides a good signoff mechanism to validate scripts, approving every new script from a nonadministrator can become laborious and unmanageable over time. To help with simplifying that burden, Jenkins 2 also supports the ability to run scripts in a Groovy Sandbox. This is enabled by checking the Use Groovy Sandbox option at the bottom of the pipeline script text window ([Figure 5-54](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_run_in_the_sand_man)).

# 

# The basic idea here is that there are a set of “whitelisted” methods maintained by Jenkins. This means that these methods are deemed to be safe to use in any script. If the option to use the Groovy Sandbox is selected and the script only makes use of methods in the whitelist that are known as safe, the script is allowed to run without approval. This saves the extra overhead of requiring an administrator to approve it.

# However, if any of the methods in the script are not in the whitelist, then the script is not allowed to run and an error is flagged ([Figure 5-55](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_meth_flag_no_go_sandbox)). In that case, the method is queued for approval by the administrator—just as the scripts are in the regular script approval process.

# 

# Here again, when the administrator logs in and goes to Manage Jenkins, they will see the alert that there is a method signature waiting for their approval

# 

# On the “In-process Script Approval” page, the administrator will be presented with a choice to Approve, Approve assuming permission check, or Deny the method ([Figure 5-57](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch05.html#fig_admn_opp_approvusemethod)). The Approve and Deny options are self-explanatory. The “Approve assuming permission check” option permits running this method if an actual user is doing it (not a system call) and assuming the user has appropriate Jenkins permissions to allow doing the operation. If approved, the method will be added to the internal whitelist.

# 