Surround SCM Best Practices

This document addresses some of the common activities in Surround SCM and offers best practices for each. These best practices are designed with Surround SCM users in mind, but many of them may apply regardless of the tool.

Organizing Files in Repositories

Source code files are stored in the Surround SCM database and organized in repositories. While you can think of repositories as a directory structure, a repository has a key difference—file history. Files in repositories are tracked over time and file changes are versioned. Who made changes, when, and why is also tracked for each file. Directories do not track this information.

Because the number of files will grow over time, it is a good idea to do some planning before you start adding files to Surround SCM.

Repositories versus branches

It is important to understand the difference between repositories and branches. A repository is used to organize files and other repositories. Users map working directories to repositories, modify files in a working directory, and check in changes to the repository. In the following example, the WysiCorp Products repository is organized by product.

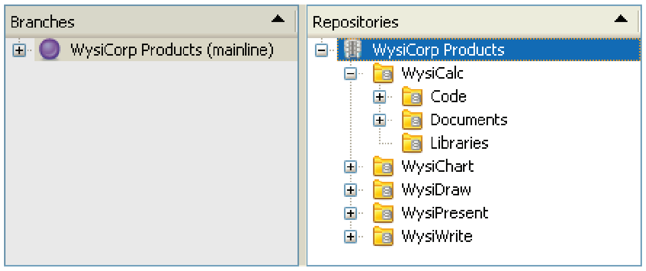


Figure 1: WysiCorp Products repositories

A branch is a separate line of development that uses an existing repository and its files as a starting point. Branching allows users to work on multiple versions of a product simultaneously while using the same set of source files. A branch represents a milestone in development. Its primary purpose is to provide a separate area for development or to represent the code at a point in time, not to store files.

The following example shows a branching structure for multiple releases. The WysiCalc 1.0.x branch was created from the WysiCalc repository on the mainline branch, which is the top-level branch that contains all repositories and branches.

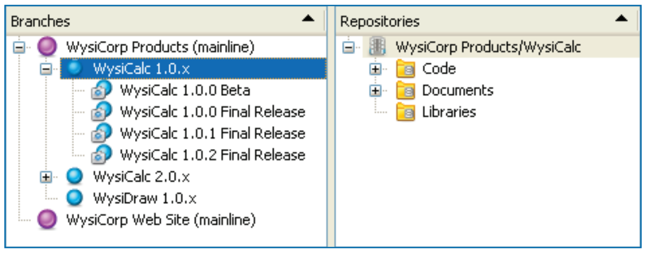

Branches
WysiCorp Products (mainline)
WysiCalc 1.0.x
WysiCalc 1.0.0 Beta
WysiCalc 1.0.0 Final Release
WysiCalc 1.0.1 Final Release
WysiCalc 1.0.2 Final Release
WysiCalc 2.0.x
WysiDraw 1.0.x
WysiCorp Web Site (mainline)
Repositories
WysiCorp Products/WysiCalc
Code
Documents
Libraries

Figure 2: WysiCalc 1.0.x branch and repositories

Do not create branches just to organize files. For example, if you are developing a new product, do not automatically add a branch for the new product. Instead, create the repository structure in the mainline branch. Perform development activities at that level until you are ready to create a release or need to begin parallel development.

When to add files to a repository

When you create a new product, you do not need to add files to the repository as soon as you create them or wait until they are complete. You can work on local copies of source code and add files to the repository when you are sure the files will not break a build or when other developers need to work with them. If your computer is not backed up regularly, or if you work remotely, you may want to add files to Surround SCM earlier to back up your work.

How to structure a repository

The easiest repository structure to maintain has a repository for each product or application on the mainline branch. Add the files and any subrepositories to each product repository. The following example shows the repository structure for a company, WysiCorp, and one of their products, WysiCalc.

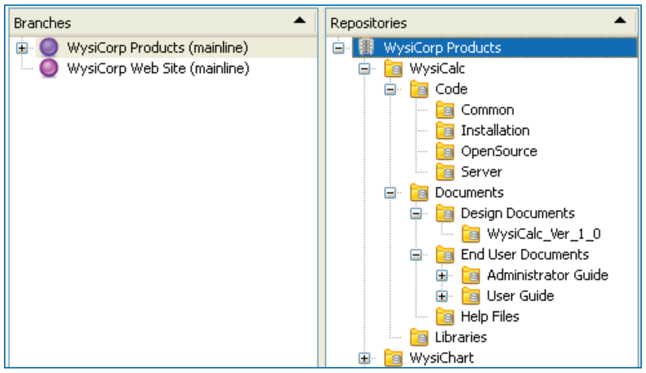


Figure 3: WysiCalc repositories

What to include in a repository

Because source code files and the file change history are stored in Surround SCM, the database can grow exponentially. In general, you only need to store the source files necessary to create a product. Do not add binary files generated during a build, such as executables and object files, to Surround SCM. You can regenerate those files if you have the source code and build scripts. This practice can save significant database space. You may want to include binary files if you need to version the files or if the time required to regenerate them outweighs the space considerations.

It is also important to store project-related documentation, including requirements, functional specifications, technical designs, and policies, in Surround SCM unless you use a requirements management tool, such as TestTrack RM. You can also store binary source files in Surround SCM. In the following example, the Documents repository contains user documentation and project documents.

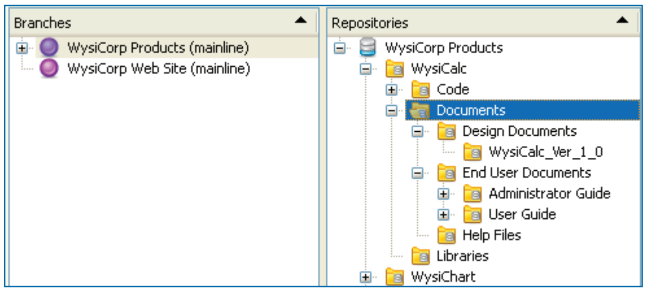


Figure 4: WysiCalc Documents repository

Branching

Branching is a balancing act. The right amount of branching can improve productivity by allowing for parallel development. Too many poorly-defined branches can cause confusion and result in unnecessary code merges. Not enough branches or resistance to branching can decrease productivity and keep you from using Surround SCM to its fullest. The following guidelines can help you achieve a balanced branching approach.

Choose the correct branch type

Surround SCM includes the following types of branches:

- **Mainline branch**–A top-level branch that contains all source files, other branches, and repositories. For most organizations, one mainline branch is sufficient. Use a single mainline branch if you will need to share files between repositories or branches.
- **Baseline branch**–A public branch used for collaborative development. Allowing check outs on a baseline branch lets all users check out and make changes to the baseline code. Changes made to files in the baseline branch affect everyone who accesses that branch.
- **Workspace branch**–A private branch that tracks changes made by an individual user. Other users are not affected by any changes made in the private workspace branches. Use workspace branches for features developed by one person.
- **Snapshot branch**–A read-only, static branch that generally corresponds to a project milestone, such as a QA build or final release. Most Surround SCM commands, such as check outs, check ins, and merges, are disabled in snapshot branches. Create a snapshot branch when you need to capture the state of the files and the repository structure at a moment in time. You can create a snapshot branch based on the latest file version, latest version in a specific workflow state, a label, or a timestamp.

Use repositories for organization and branches for parallel development

After you set up the repository structure on the mainline branch, use branching to control releases or for parallel development. Do not create repositories when you mean to create a branch. For example, when you are ready to release a product and start development on a new version, create a branch for the release code.

The following example shows the correct use of branches and repositories. The WysiCalc 1.0.x branch is created from the WysiCalc repository in the WysiCorp Products mainline branch. When the branch is created, the repositories and files in the WysiCalc repository on the mainline are copied to the new branch. Developers can start working on WysiCalc 1.0.x without affecting the stable code base in the mainline branch.

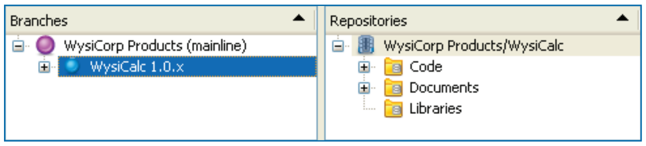


Figure 5: Correct branch and repository use

Branching by purpose

While the ideal branching model depends on your business needs, we recommend using a branch by purpose model. In this model, you create a branch when you need to provide the software to groups outside of the development team. You create a branch for alpha testing or a product release. Branching by purpose also supports creating a branch for research and development. You can create a branch for developers to use when evaluating changes to the code without affecting the stable code base. The following example shows branching by purpose.

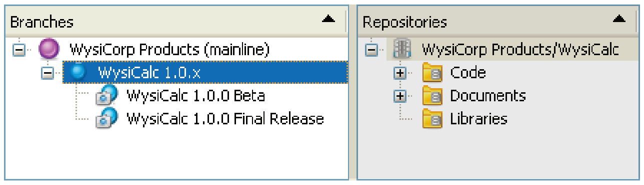

Branches
WysiCorp Products (mainline)
WysiCalc 1.0.x
WysiCalc 1.0.0 Beta
WysiCalc 1.0.0 Final Release
Repositories
WysiCorp Products/WysiCalc
Code
Documents
Libraries

Figure 6: Branch by purpose example

After all WysiCalc features are complete, WysiCorp creates the WysiCalc 1.0.x baseline branch. Some developers continue working on the WysiCalc 1.0.x branch, fixing bugs and preparing for release, while other developers work on the next version of the software on the mainline branch. When the code is ready for testing, the WysiCalc 1.0.0 Beta snapshot branch is created to document the tested build.

The developers working on the WysiCalc 1.0.x branch can fix bugs found during beta testing and promote them to the mainline branch so the developers working on the next release can incorporate them into the latest code. When the code is ready for release, the WysiCalc 1.0.0 Final Release snapshot branch is created. The developers can continue performing maintenance on the WysiCalc 1.0.x branch to fix production-level defects and create maintenance releases without affecting the parallel development of version 2.0.

Branching late

Branching late goes hand-in-hand with branching by purpose. In the branch by purpose model, developers work on the mainline branch until it is time to provide code to another group. In the previous example, developers worked on the mainline branch until alpha testing started and the WysiCalc 1.0.x branch was created.

Branching late also reduces the number of merges to perform. Using the previous example, instead of continuing to work on the mainline branch to develop WysiCalc version 2, a WysiCalc 2.0.x branch was created for version 2.0.x development at the same time the WysiCalc 1.0.x branch was created.

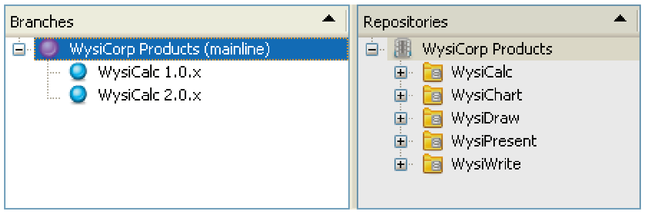


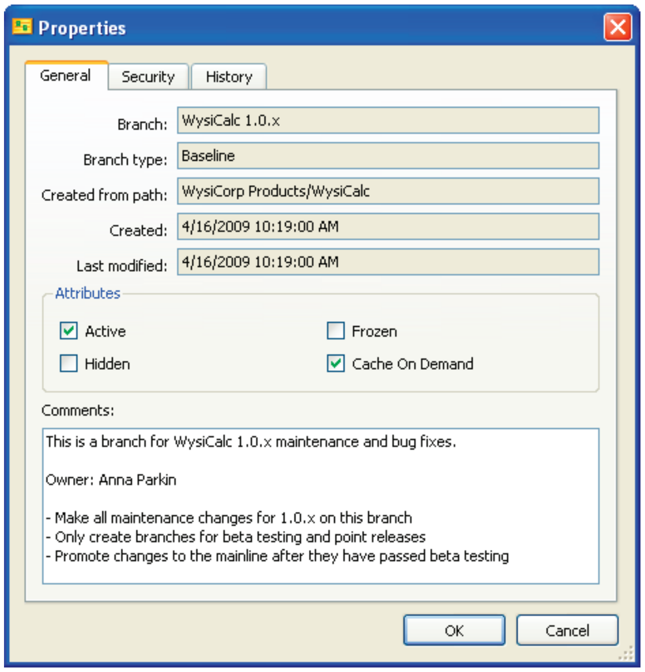
Figure 7: Branch late example

In this case, the developers will have to integrate changes from two branches into the mainline branch from the beginning of development on version 2. By branching late, they only need to integrate changes from one branch into the mainline until they are ready to begin testing version 2, simplifying the merge process.

Create a branching policy

If multiple users will be creating branches, you should create a branching policy that specifies when to branch, how to name branches, how to document branches, who can create branches, and your sharing strategy. As a general rule, development managers and release managers should be able to create baseline branches, while developers should be able create their own workspace branches. Store the branching policy document in the top-level repository on the mainline branch so everyone can access it. Also, create a trigger that emails developers when the document changes.

When you create a branch, use comments to explain the purpose of the branch, when changes should be checked in or out, when changes should be promoted and merged, and who owns the branch. As the number of branches grows, defining branch ownership can help developers sort out issues and avoid confusion.



Branch: WysiCalc 1.0.x
Branch type: Baseline
Created from path: WysiCorp Products/WysiCalc
Created: 4/16/2009 10:19:00 AM
Last modified: 4/16/2009 10:19:00 AM

Attributes
- [✓] Active
- [ ] Frozen
- [ ] Hidden
- [✓] Cache On Demand

Comments:

This is a branch for WysiCalc 1.0.x maintenance and bug fixes.

Owner: Anna Parkin

- Make all maintenance changes for 1.0.x on this branch
- Only create branches for beta testing and point releases
- Promote changes to the mainline after they have passed beta testing

Figure 8: A well-documented branch

Add new repositories to the mainline branch

When you set up Surround SCM or add files for a new project or product, add the repository and files to the mainline branch. This keeps the original source code on the mainline branch and allows you implement branches for specific development purposes from the mainline. Develop on the mainline branch until there is a need to create a branch.

After you establish branches, it is best to add new files and repositories to the mainline branch and rebase to propagate the new files and repositories to the other branches.

Checking Files In and Out

Create a check in/check out policy

Like the branching policy, a check in/check out policy ensures developers are working as efficiently as possible. Your policy should specify the following:

- When developers should use exclusive or non-exclusive (multi-user) check outs
- How often to check in files
- Who is responsible for performing merges (individual developers, lead developers, project manager, etc.)

Store this policy in the root repository of the mainline branch and create a trigger that emails developers when the document changes.

Check in frequently

Regardless of the type of check outs you use, check files in frequently, such as when you complete a set of changes or when you need to share changes with other developers. Make sure you do not check in code that may not compile or that may break a build.

Checking in regularly has several benefits. If you are using non-exclusive check outs, it is necessary to merge more frequently. Checking in regularly (daily or weekly) reduces the complexity of these merges and makes it easier to resolve conflicts. Checking in often also provides access to changes made by other developers and prevents surprises later in development.

Provide detailed comments on check in

You can enter comments when you check files into Surround SCM. Your changes and the logic behind them might be clear now, but will you remember why you made the change two months from now when you are in the middle of another fix?

Include the following information when you check in files:

- The original cause for the change (bug or enhancement)
- The bug or defect number fixed
- The intention of the change
- The methodology used to fix the bug or create the enhancement and any other alternatives you considered

Grouping Related Changes

Use changelists to group transactions

A changelist, which is a set of files and the actions to be performed on those files, groups transactions together and treats the files as one unit. Changelists provide an easy way to track the files that were changed to fix a defect.

Changelists implement atomic transactions–if one action in a changelist fails, the entire operation is cancelled and changes are not committed to the database. Users can also view the history of changelists and see which files were processed together.

You can add files to a changelist when you add a file, check in a file, create a repository, remove a file or repository, or rename a file or repository.

Attach file changes to related issues

If you make file changes to address a defect or issue tracked in TestTrack or another tool, attach the changed file to the associated issue during check in. This helps other team members clearly understand the files that were changed to fix an issue, whether they are looking at the changes from Surround SCM or the issue tracking tool. For example, attaching files to issues can help release managers understand which files to get in the build that includes the fix, QA managers understand areas that require testing based on files that changed, and development managers see which files need to go through code review.

Apply labels to group related files in different repositories

Labels are text tags that mark a file version and make it easy to retrieve versions marked for a specific reason, such as a defect fix, build, or release. Use labels if you need to group related files for future reference, especially if they are stored in different repositories in a branch.

You can apply labels when add files, get files, check in files, promote files or branches, duplicate changes to a branch, view a committed changelist, or view file history.

Merging Branches

Promote or rebase to sync files between related branches

Promoting and rebasing updates related branches with changes made in a selected branch, ensuring the related branch includes the most current files and that other users have access to the changes. All changes made to a file starting at version 1 until a point you choose, such as the latest version or a timestamp, are merged to the related branch.

Promoting merges changes from the child branch up to the parent branch. Promote changes regularly to make sure the latest fixes are available on the mainline branch. The promotion process resembles the following example.

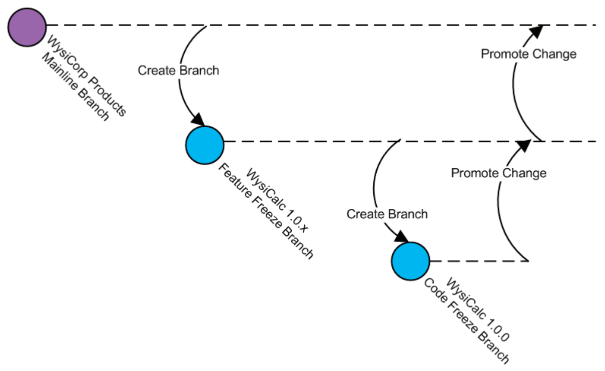

WysiCorp Products Mainline Branch
Create Branch
Promote Change
WysiCalc 1.0.x Feature Freeze Branch
Create Branch
Promote Change
WysiCalc 1.0.0 Code Freeze Branch

Figure 9: *Promotion example*

At the feature freeze milestone, which is when feature development for the next WysiCalc release is complete, WysiCorp creates a baseline branch named WysiCalc 1.0.x from the WysiCalc repository on the mainline branch.

Feature development for the next major release continues in the WysiCorp Products mainline branch while other developers fix defects in the WysiCalc 1.0.x branch. A code freeze branch named WysiCalc 1.0.0 is created after deciding that WysiCalc is ready for release. Critical defect fixes for the release are made in the WysiCalc 1.0.0 code freeze branch and promoted to both the WysiCalc 1.0.x and WysiCorp Products branches.

Rebasing merges changes from the parent branch down to the child branch. For example, if two developers are working on bug fixes in workspace branches and one developer completes work and promotes changes to the parent branch, the other developer can rebase the changes into their workspace to make sure they are working with the most current files.

Define the promote and rebase policy as part of your branching policies.

Duplicate to merge specific changes between any two branches

If you want to choose a set of changes made in one branch and apply them to another branch, duplicate the changes instead of promoting or rebasing. When you duplicate changes, you choose the changes between two versions to merge into another branch. Changes made in other versions are ignored. You can also duplicate changes by changelist, defect, or label. You can duplicate changes to any other branch in the same mainline branch, not only to a parent or child branch like promoting and rebasing.

For example, if a developer makes changes in a workspace branch that fixes a defect related to two different products, she can duplicate the changes in files attached to the defect to each product’s baseline branch without unnecessarily merging all other changes made in her workspace branch to other related branches.

Capturing Milestones

Indicating the code included in builds and releases is a critical step for maintaining an accurate history of your software development effort. You can capture these milestones using labels and snapshot branches.

Use snapshot branches to capture important builds and releases

A snapshot branch is a read-only, static branch. Create snapshot branches after successfully completing important builds or releases. You can create snapshot branches based on the latest file version, the latest version in a specific workflow state, a label, or a timestamp.

Snapshot branches make it easy to re-create builds because, unlike labels, they capture the filenames and repository structure used when the branch was created. Labels only capture file content and builds may break if files are renamed or moved in Surround SCM after successfully completing a build.

Use labels to capture daily builds

If your organization performs daily builds, or even multiple builds per day, use labels to capture the files included in the build. Using labels makes it easy to tag the files included in a build and retrieve them for future use. They also allow you to preserve a history of a build without adding the clutter of multiple snapshot branches in the branch tree.

Labels also offer more flexibility than snapshot branches. You can add files to a label as you work with files or after creating a build. You can also replace files in a label with other versions over time. If you use snapshot branches, you cannot make changes after creating the branch.

Using Triggers

You can use triggers to perform actions before or after a specific event. Triggers can be used for notifications, validation, custom text entry, logging, and synchronization.

Pre-event triggers

Pre-event triggers run after a client requests that an event be performed, but before the event is complete. These triggers can be used to run server-side executables (scripts and compiled programs), prevent users from performing file events and display a message that explains why the event cannot be performed, and add comments or append text to comments that users enter on file events.

Use pre-event triggers with caution because they run once per file and can cause performance issues because the Surround SCM Server waits for the script to complete before moving to the next file. Even a one second pre-event script will significantly slow down the server because it takes an additional second per file that the trigger fires on.

Post-event triggers

Post-event triggers run after an event is successfully completed on the Surround SCM Server. These triggers can run server-side executables, send emails, change custom field values, and change workflow states. Email triggers use an email template that can be customized to inform selected users when an event occurs to a file or a set of files. Post-event triggers should be used unless the trigger is used for validation.

Configure server access

Because triggers are performed on the server, any scripts they execute must also be stored on the server. Make sure the processes the triggers call have the right server permissions to run correctly.

Administering Surround SCM

Use the recommended hardware configurations

To avoid performance issues, use the recommended hardware configurations in the Surround SCM Server System Requirements knowledgebase article (www.seapine.com/kb/questions/1173).

Set proper file-level permissions

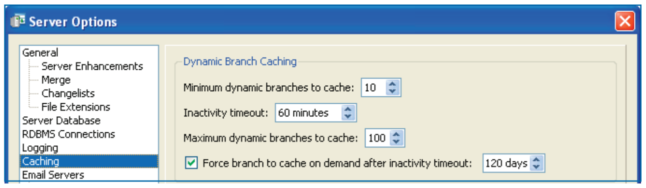
Users do not need read and write access to the Surround SCM database. Only the Surround SCM Server requires read/write access to the database. All access rights are controlled through the Surround SCM Client. This adds an additional level of security and prevents users from making changes directly in the database and bypassing version control.

Adjust caching options to improve performance

Over time, the number of branches in Surround SCM will increase and you will not access all branches with the same regularity. There will be branches that developers access every day and others that are essentially archives. If you start to have performance issues, you may need to adjust the number of branches stored in the Surround SCM Server cache.

Branches are cached when the Surround SCM Server starts. By default, mainline branches are set to Always Cache. Baseline, snapshot, and workspace branches are set to Cache on Demand. Cache on Demand, or dynamic, branches are only cached when users perform actions that access the branch. The most recently used dynamic branches as of the last server shutdown are cached at server startup.

Surround SCM automatically manages the cache by unloading dynamic branches and changing Always Cache branches to Cache on Demand based on inactivity. If you notice memory issues and slow server start up times, adjust the server options to control the cache size and inactivity timeout limits.



Dynamic Branch Caching

Minimum dynamic branches to cache: 10

Inactivity timeout: 60 minutes

Maximum dynamic branches to cache: 100

✓ Force branch to cache on demand after inactivity timeout: 120 days

Figure 10: *Surround SCM Server caching options*