Overview

The Rock branching strategy is based on the Git Branching Model documented by Vincent Driessen.

Branches

Master

The `master` branch should always reflect the latest production-ready state, and should be tagged with the appropriate release number. The only time the master branch is updated is when a `release` or `hotfix` branch is merged into `master`, at which point it is tagged with the appropriate version number. Development should never occur from the `master` branch.

Develop

The `develop` branch is always the latest runnable code. A developer should always be able to pull the latest version of the `develop` branch, run an update-database to create a test database, and then be able to code and test against this.

When a developer needs to make an update to the source code, they need to decide whether to create a `feature` branch (documented below), or work directly off their local version of the `develop` branch. If there's any chance that while working on a specific code change, they may need to make a separate unrelated change, then they should create a local `feature` branch rather than working directly from the `develop` branch. If making a small change that will be committed/pushed quickly, then developers can work directly off of their local `develop` branch.

A good rule of thumb, is that if a change requires more than a day's time to develop and test, then it should probably be done in a `feature` branch. If it can be completed within a day, then it can be done on the `develop` branch

Feature

A `feature` branch is used to `develop` new features for a future release. Feature branches should be branched from develop and eventually merged back into `develop`. The naming convention for feature branches is `feature-[Initials]-[name-of-feature]`. For example `feature-drt-add-campus`. Typically, feature branches would only exist in the developers repository, however, if more than one developer are working on (or reviewing) a specific feature, that feature branch can be pushed to the remote origin repository (GitHub). Once a feature has been completed, the feature branch is deleted from both the developer's repository and from the remote repository.

Release

A `release` branch is created when development has been completed for a particular target release. The naming convention for `release` branches is `release-[version-number]`. For example, `release-1.0`. Release branches are branched from the `develop` branch and will be created in both the remote repository, and in the local repository of any developer working on the release.

Once the release is ready to be distributed, the `release` branch is merged into the `master` branch, and the master branch is tagged with the release number. If any changes were made to the `release` branch after it was branched from `develop`, it will then also need to be merged back to `develop`. The `release` branch is then deleted from both local and the remote repository.

Hotfix

A `hotfix` branch is created when an issue is discovered for the latest release. Hotfix branches are similar to `release` branches in naming convention and how they are treated when complete. The difference is that they are branched from the `master` branch rather than the `develop` branch. Naming convention is `hotfix-[version-number]`. For example `hotfix-1.0.1`.

Just like a `release` branch, once development and testing are complete for the hotfix, it is merged into the `develop` and the `master` branch, which is then tagged with the updated version number. The `hotfix` branch is then deleted from both the remote and local repositories.

Support

A `support` branch is created when a critical bug is found on a previous version of the application and a customer using that version cannot upgrade to the most recent version. A `support` branch will be branched from the `master` branch at that version's tag on the `master` branch. For example, if the latest released version number is 1.0 and a bug is found in version 1.0, a `hotfix` branch rather than a `support` branch would be created. That `hotfix` branch is then merged back into `master`, and `master` would be tagged 1.0.1. If an issue is found with the 1.0.1 version after version 2.0 has already been released (and merged into `master`), then a `support` branch would be created by branching from the 1.0.1 tag position in the `master` branch. That `support` branch is never merged back into any other branch. It remains as long as that version of the application needs to be supported. Once the change has been made to the `support` branch and tested and released, it should be tagged with an appropriate version number (i.e. 1.0.2). If further support needs to be done on a previous release, a `hotfix` branch could be branched from the `support` branch and then merged back into the `support` branch, similar to how `hotfix` branches are used on the most recent version on the `master` branch.

Custom

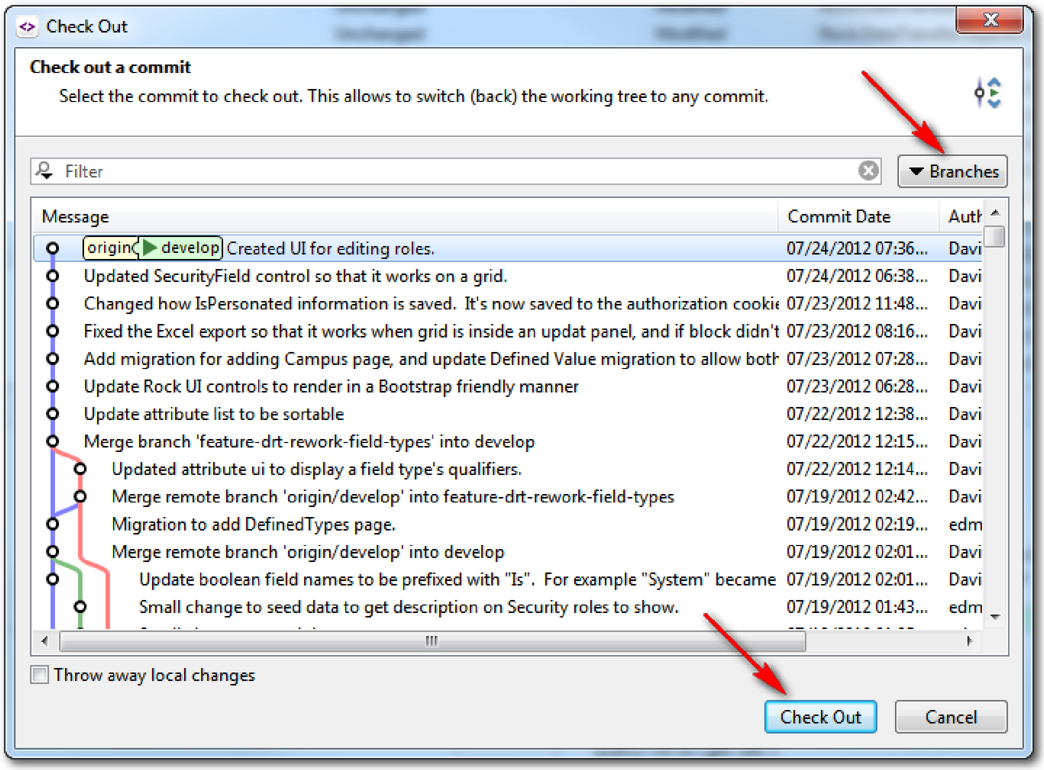
Custom branches are special branches used by the organizations that are contributing to the development of the application and includes code/data that is specific to their organization. The naming convention for a `custom` branch is `custom-[organization domain]`. For example `custom-ccvonline`. It is up to each of those organizations to determine how their branch is managed and how and when code from the `develop` and/or `master` branches is merged into their specific `custom` branch.

Using SmartGit

New Branch

It is easiest if you switch to the branch that you would like to branch from. For example if creating a new release branch, switch to the develop branch.

Select *Branch -> Check Out* from the menu. Make sure that the correct branch/commit is selected in the list of commits. You can filter the list of branch/commits based on selected branches using the “Branches” button in the top right part of the screen. Once you’ve selected the correct commit, click the “Check Out” button.



Check Out

Check out a commit

Select the commit to check out. This allows to switch (back) the working tree to any commit.

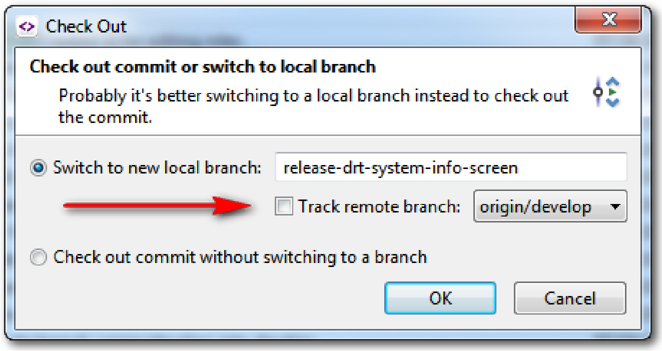
Filter | Branches

Message	Commit Date	Auth
origin develop Created UI for editing roles.	07/24/2012 07:36...	Davi
Updated SecurityField control so that it works on a grid.	07/24/2012 06:38...	Davi
Changed how IsPersonated information is saved. It's now saved to the authorization cookie	07/23/2012 11:48...	Davi
Fixed the Excel export so that it works when grid is inside an updat panel, and if block didn't	07/23/2012 08:16...	Davi
Add migration for adding Campus page, and update Defined Value migration to allow both	07/23/2012 07:28...	Davi
Update Rock UI controls to render in a Bootstrap friendly manner	07/23/2012 06:28...	Davi
Update attribute list to be sortable	07/22/2012 12:38...	Davi
Merge branch 'feature-drt-rework-field-types' into develop	07/22/2012 12:15...	Davi
Updated attribute ui to display a field type's qualifiers.	07/22/2012 12:14...	Davi
Merge remote branch 'origin/develop' into feature-drt-rework-field-types	07/19/2012 02:42...	Davi
Migration to add DefinedTypes page.	07/19/2012 02:19...	edm
Merge remote branch 'origin/develop' into develop	07/19/2012 02:01...	Davi
Update boolean field names to be prefixed with "Is". For example "System" became	07/19/2012 02:01...	Davi
Small change to seed data to get description on Security roles to show.	07/19/2012 01:43...	edm

☐ Throw away local changes

Check Out | Cancel

Then enter the name for your new branch, and UNSELECT the “Track remote branch” option. In all of the SmartGit dialog windows, this is the only place where we don’t use the default option.



Check Out

Check out commit or switch to local branch

Probably it's better switching to a local branch instead to check out the commit.

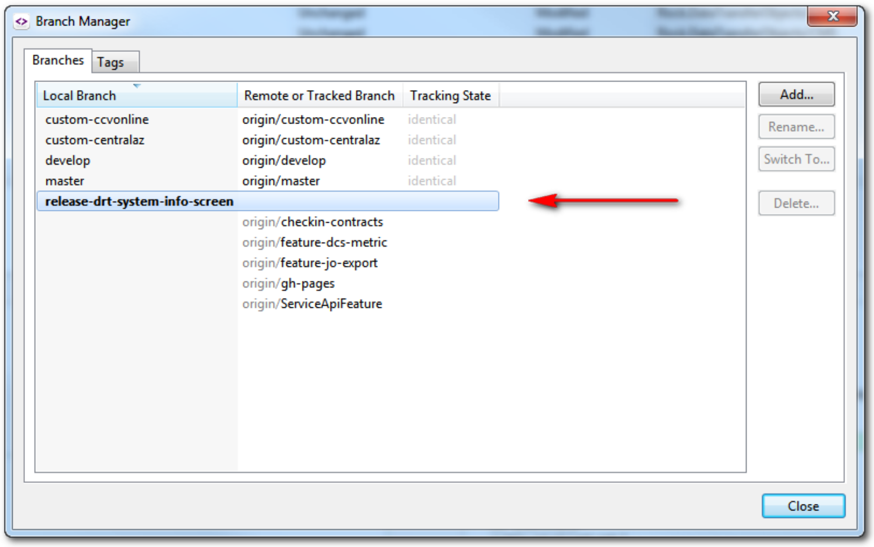
[✓ Switch to new local branch:] release-drt-system-info-screen

☐ Track remote branch: origin/develop

Check out commit without switching to a branch

OK | Cancel

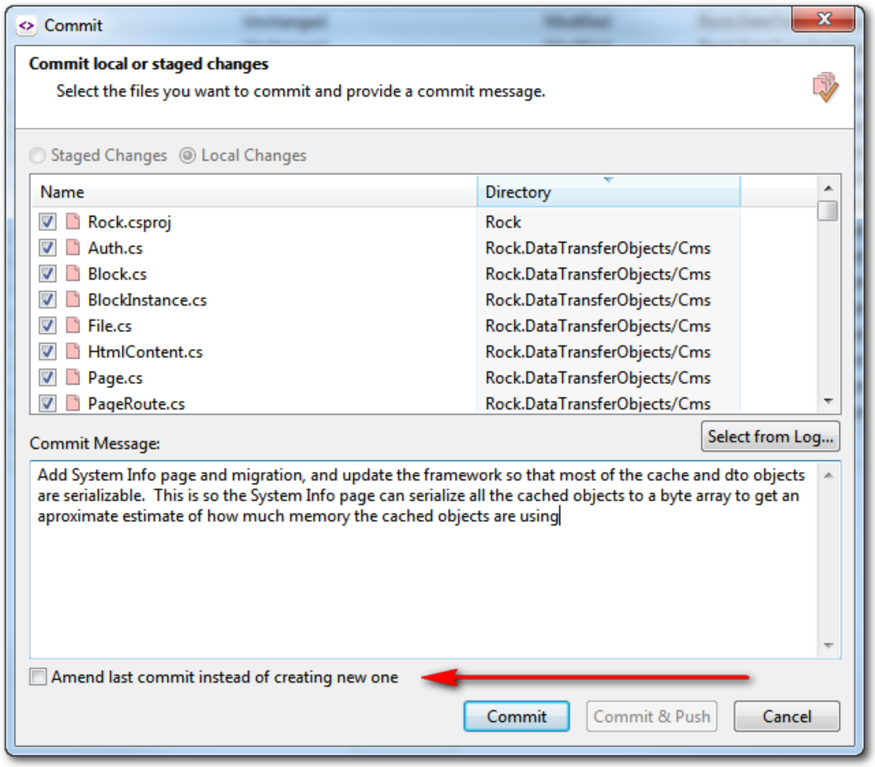
This creates a new branch in the local repository. You can view all the local and remote branches by using the *Branch -> Branch Manager* menu option…



Local Branch	Remote or Tracked Branch	Tracking State
custom-ccvonline	origin/custom-ccvonline	identical
custom-centralaz	origin/custom-centralaz	identical
develop	origin/develop	identical
master	origin/master	identical
release-drt-system-info-screen		
	origin/checkin-contracts	
	origin/feature-dcs-metric	
	origin/feature-jo-export	
	origin/gh-pages	
	origin/ServiceApiFeature	

Committing Changes

Once you've created a new branch, it is important to commit changes often. You do have the option of combining a commit with the previous commit by selecting the "Amend last commit instead of creating new one" option…



Commit local or staged changes

Select the files you want to commit and provide a commit message.

Name	Directory
✓ Rock.csproj	Rock
✓ Auth.cs	Rock.DataTransferObjects/Cms
✓ Block.cs	Rock.DataTransferObjects/Cms
✓ BlockInstance.cs	Rock.DataTransferObjects/Cms
✓ File.cs	Rock.DataTransferObjects/Cms
✓ HtmlContent.cs	Rock.DataTransferObjects/Cms
✓ Page.cs	Rock.DataTransferObjects/Cms
✓ PageRoute.cs	Rock.DataTransferObjects/Cms

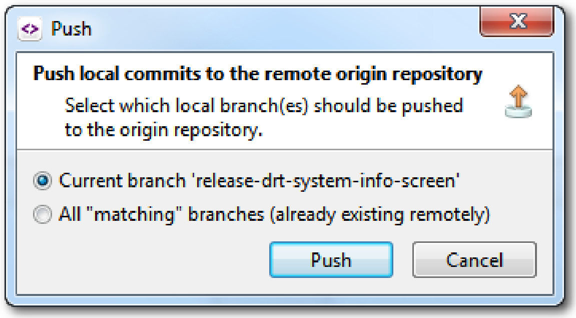
Commit Message:

Add System Info page and migration, and update the framework so that most of the cache and dto objects are serializable. This is so the System Info page can serialize all the cached objects to a byte array to get an aproximate estimate of how much memory the cached objects are using

☐ Amend last commit instead of creating new one

Pushing Branch

To push a branch to the remote repository (GitHub) for other developers to see or work on, you simply do a push when viewing the branch. Select the "Push" button…

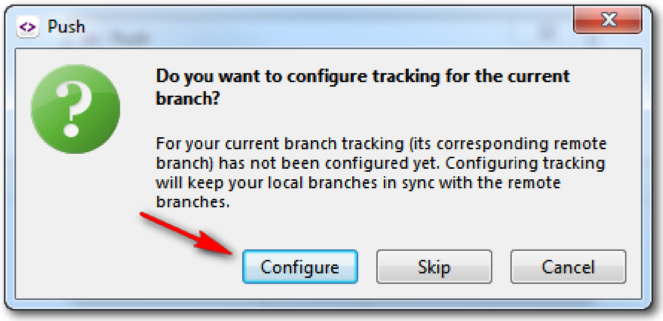


Push local commits to the remote origin repository

Select which local branch(es) should be pushed to the origin repository.

[✓ Current branch 'release-drt-system-info-screen']
All "matching" branches (already existing remotely)

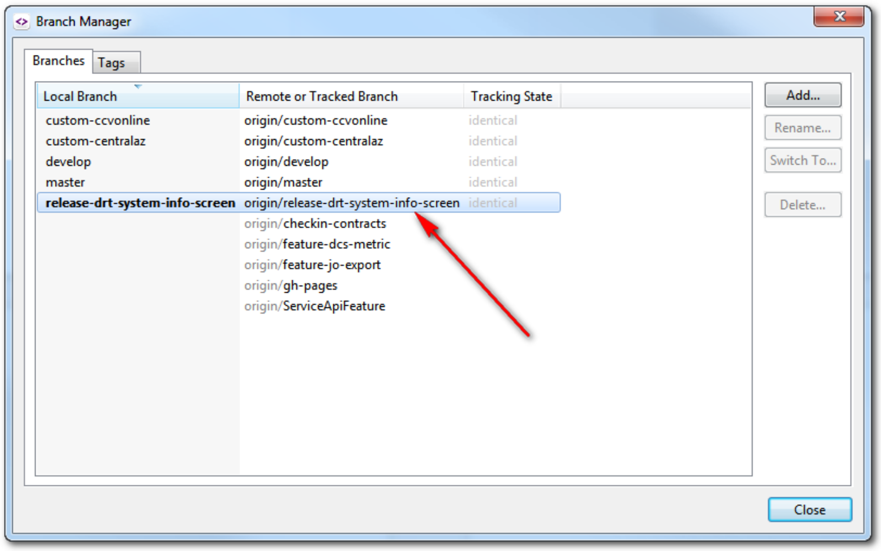
The first time you push to the remote repository, SmartGit will ask if you'd like to configure tracking for the remote branch. You should select the "Configure" option…



Do you want to configure tracking for the current branch?

For your current branch tracking (its corresponding remote branch) has not been configured yet. Configuring tracking will keep your local branches in sync with the remote branches.

Now the branch will also be on the remote repository for other developers to pull to their local repository…



Local Branch	Remote or Tracked Branch	Tracking State
custom-ccvonline	origin/custom-ccvonline	identical
custom-centralaz	origin/custom-centralaz	identical
develop	origin/develop	identical
master	origin/master	identical
release-drt-system-info-screen	origin/release-drt-system-info-screen	identical
	origin/checkin-contracts	
	origin/feature-dcs-metric	
	origin/feature-jo-export	
	origin/gh-pages	
	origin/ServiceApiFeature	

Switching Branch

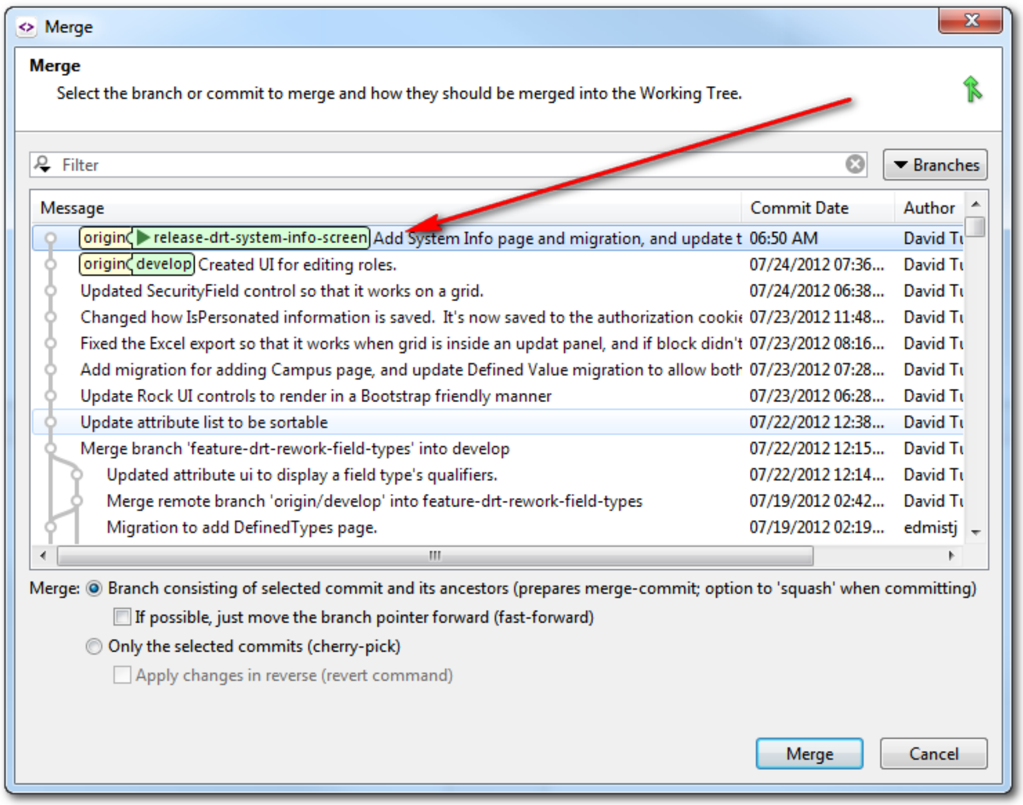
Prior to switching branches, you should commit any uncommitted work that has been made for the current branch. Git will attempt to update your working copy with all the committed changes for the branch that you switch to. If you have uncommitted changes, You will likely get an error from Git that will prevent you from switching.

To switch branches, select the *Branch -> Switch* menu option or the Switch button in the toolbar. This will display the branch dialog and you can choose the branch to switch to. Again, Git will update your working copy to reflect the committed state of the branch that you switched to.

Merging

To merge one branch into another, first switch to the branch that you would like to merge to. For example when merging changes from a `feature` branch, first make sure all the changes are committed in that branch, then switch to the `develop` branch.

Select the B*ranch -> Merge* menu option, or the Merge toolbar button. From the Merge dialog window, select the branch/commit that you'd like to merge from…



Merge

Select the branch or commit to merge and how they should be merged into the Working Tree.

Message	Commit Date	Author
origin release-drt-system-info-screen Add System Info page and migration, and update t	06:50 AM	David Tu
origin develop Created UI for editing roles.	07/24/2012 07:36...	David Tu
Updated SecurityField control so that it works on a grid.	07/24/2012 06:38...	David Tu
Changed how IsPersonated information is saved. It's now saved to the authorization cookie	07/23/2012 11:48...	David Tu
Fixed the Excel export so that it works when grid is inside an updat panel, and if block didn't	07/23/2012 08:16...	David Tu
Add migration for adding Campus page, and update Defined Value migration to allow both	07/23/2012 07:28...	David Tu
Update Rock UI controls to render in a Bootstrap friendly manner	07/23/2012 06:28...	David Tu
Update attribute list to be sortable	07/22/2012 12:38...	David Tu
Merge branch 'feature-drt-rework-field-types' into develop	07/22/2012 12:15...	David Tu
Updated attribute ui to display a field type's qualifiers.	07/22/2012 12:14...	David Tu
Merge remote branch 'origin/develop' into feature-drt-rework-field-types	07/19/2012 02:42...	David Tu
Migration to add DefinedTypes page.	07/19/2012 02:19...	edmistj

Merge: [✓ Branch consisting of selected commit and its ancestors (prepares merge-commit; option to 'squash' when committing)]

☐ If possible, just move the branch pointer forward (fast-forward)

Only the selected commits (cherry-pick)

☐ Apply changes in reverse (revert command)

All of the changes from the branch will now be displayed as uncommitted changes to the current branch. There may be some files with conflicts if Git could not figure out how to merge changes from the source branch into the target branch. In this case you will need to look at each conflicted file and choose the appropriate change. Once you've saved the change, SmartGit will ask you if the change would be staged (select yes). Once you've resolved any conflicts, commit all the changes, and then if necessary do a Pull and Push to update the remote repository.