# Using Git

We use Git to manage our source code. Our Stash installation is the front end we use to manage our different Git repositories.

This page will help you get started with Git and getting the KIELER sources. For more detailed information, see Git's official documentation. The SVN Crash Course is probably a good place to start. For more in-depth information, see the Git Community Book or the Pro Git book. Furthermore, each office has a copy of another excellent book about Git, so you might as well go ahead and read it. This will help ease you in to some of the more advanced concepts of Git, which are a little hard to understand at first. If everything else fails, Miro and Tim will be more than happy to help you with Git and rant about how excellent of a system it is. 😉 For more information on Git Eclipse integration, see the EGit User's Guide.

**Content**

## Checking Out KIELER

You shouldn't need to check out anything manually if you have installed Eclipse using our Oomph setup.

KIELER is essentially a large heap of Eclipse plug-ins that aren't easy to find your way through as a new developer. The Overview page has a nice overview of our sub-projects and what plug-ins belong where. This section will tell you how to get the KIELER sources. As for what plug-ins you will actually need to checkout, ask your advisor.

**Important Hint for Users Behind Firewalls**

Repository access via SSH runs on port 7999. For accessing the repositories in read-only mode, HTTP transfer is also possible, but not recommended.

Checkout of the Git repository is possible either using the SSH or the HTTP protocol. We strongly recommend using SSH; if you still want to use HTTP, omit the SSH key creation and upload in the instructions below.

1. If you don't have an SSH key yet, you have to create one. You can do this by:
    - Creating one using the command `ssh-keygen` on the command line. Simply type `ssh-keygen`, confirm the default destination file ~/.ssh/id_rsa, and choose whether to give a passphrase. If you have a passphrase, you need to enter it whenever you use your SSH key for the first time in a session. You can omit the passphrase, but that makes the key less secure. As result, the tool generates a private key ~/.ssh/id_rsa, which has to be kept secret, and a public key ~/.ssh/id_rsa.pub.
    - Using Eclipse to generate it. You can find this function under *Preferences - General - Network Connections - SSH2 - Key Management*.
2. Register with Stash and upload your public SSH key (*Profile - SSH Keys - Add Key*).
3. Copy one of the following repository URIs into the clipboard: `ssh://git@git.rtsys.informatik.uni-kiel.de:7999/KIELER/the_repo_to_clone.git` where `the_repo_to_clone` is either `pragmatics` or `semantics` (if in doubt, ask your adviser which of these you need). If you for whatever reason insist on using the less efficient HTTP protocol, use the following URI: `http://youraccountname@git.rtsys.informatik.uni-kiel.de/scm/KIELER/the_repo_to_clone.git`)
4. Open the *Git Repositories* view, right-click it, select *Paste Repository Path or URI*, select *ssh* connection protocol, *Next*, select master branch, *Next*, select destination directory (e.g. /home/<username>/shared/kieler), *Finish*. Wait for the repository to be downloaded to your computer. Note that the whole history of the repository will be stored in your local filesystem, which is pretty awesome.
5. Right-click the *Working directory* entry in the *kieler* repository, select *Import Projects*, *Next*, select the projects that you want in your workspace, *Finish.*

Checking out on the command line is done with the command `git clone <URI> <local path>`. Instead of the URI you can also use a path to a local repository, which then creates a clone of that repository.

### Adding an Existing Local Repository to EGit

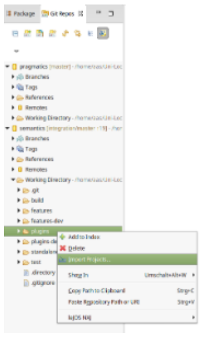
If you have already cloned the KIELER repository and are only looking for a way to import it into EGit, follow these steps:

1. Click the button *Add an existing local Git Repository to this view* in the *Git Repositories* view and enter the local path.

2. Import the plugin projects that you need.

## Importing Plugins to the Eclipse Workspace

In the Git Repository View, perform **Right-click > Import Projects...** on the *Working directory* or *Plugins* folder. Hit *Next*. In the following dialog you can *Deselect all* and afterwards select the plugins that you need for your developing task.



For example, if you want to start **KIELER with SCCharts** you need to import

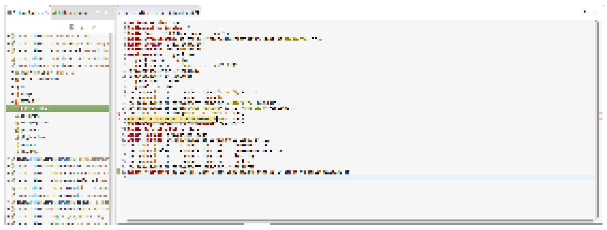
1. **all core plugins** (core.*) from the semantics and pragmatics repositories
2. **all SCCharts plugins** (sccharts.*) from the semantics repository, **unless sccharts.prio.dependencies**, **sccharts.prio.dependencies.klighd**, **sccharts.prio.s**, **sccharts.prio.sim.s**
3. **all required plugins** for the already imported ones

As a result you will have the following plugin projects in your workspace:

1. **From** the **pragmatics repo**: core, core.kgraph, core.kgraph.text, core.kgraph.text.ui, core.kivi, core.krendering, core.krendering.extensions, core.ui, kiml, kiml.formats, kiml.graphviz.dot, kiml.graphviz.layouter, kiml.service, kiml.ui, klay.layered, klighd, klighd.piccolo, klighd.ui, edu.umd.cs.piccolo
2. **From** the **semantics repo**: core.annotations, core.annotations.edit, core.annotations.text, core.annotations.text.ui, core.kexpressions, core.kexpressions.edit, core.kexpressions.keffects, core.kexpressions.keffects.edit, core.kexpressions.keffects.ui, core.kexpressions.text, core.kexpressions.text.ui, core.kexpressions.ui, core.model, core.perspectives, core.product, kex, kex.ui, kico, kico.klighd, kico.ui, kitt, kitt.klighd, prom, s, s.sc, s.sim, s.sim.kivi, s.sim.sc, s.sim.sj, s.sj, s.ui, sc, sccharts, sccharts.edit, sccharts.editor, sccharts.eso, sccharts.kivi, sccharts.klighd, sccharts.prom, sccharts.s, sccharts.scg, sccharts.sim.c, sccharts.sim.s, sccharts.text, sccharts.text.ui, scg, scg.s, scl, sim.benchmark, sim.instructions, sim.kiem, sim.kiem.config, sim.kiem.ui, sim.kiem.ui.datacomponent, sim.kivi, sim.signals, sim.signals.ui, sim.syncsignalreset, sim.table, sjl, org.freemarker, org.json

### Troubleshooting / Resolving Plugin Dependencies

If there are errors in your workspace, they are most likely the result of missing plugins. To solve this, check if the MANIFEST.MF file of the project has error markers. **Import missing plugin dependencies** if required.



After all dependencies are solved and there are still errors, you should clean your workspace via **Project > Clean > All projects**.

If there are errors in an xtend-gen folder you can delete this folder so that the contents are re-compiled. (It sometimes happens that this folder is not deleted as part of the clean.)

There should not be any errors after all required plugins are imported and compiled correctly.

## Updating the Repository

Your working copy must be clean before you can merge any updates into it. Therefore, always commit your changes locally before you pull. If you don't want to commit them into the master branch, commit them into a new branch. Note that pulling is the same as fetching remote changes and merging them into your local branch. Since a normal merge operation is involved, this can lead to conflicts, which need to be resolved as described below. Note that pulling always merges the remote changes into your current branch. If that's not what you want, checkout the correct branch first or just do a *fetch*.

Do one of the following three things:

- Right-click one of the KIELER projects, then from the *Team* submenu choose *Pull*.
- Right-click the KIELER repository in the *Git Repositories* view and click *Pull*.

- Enter `git pull` on the command line and refresh your workspace.

## Resolving Conflicts

Whenever branches are merged or rebased, conflicts can occur. They can be resolved by adjusting the state of your working copy, marking it as clean, and committing the changes. Conflicted files are modified so they contain both your version and the remote version. Edit them until all conflict areas are clean. In Eclipse this can be done by right-clicking the conflicted files - *Team - Merge Tool*.

If you try to pull from a remote repository or merge branches although you have local uncommitted changes, and the merge would affect the same files, Git leaves those files as they are, thus discarding the changes that would be made by the pull or merge operation. As a consequence, committing such a conflicted state would ignore the changes of the merged branch even if the commit graph looks like a regular merge has happened. Therefore keep in mind never to pull or merge with uncommitted changes.

**Taking your version of conflicted files:** `git checkout --ours <path>`

**Taking the remote version of conflicted files:** `git checkout --theirs <path>`

## Committing Changes

Git manages changes in two separate steps:

1. Commit changes to your local repository.
2. Push the commit to the remote repository.

The first step can be done offline, which is very useful in situations where you don't have an internet connection, but would like to save intermediate development snapshots in the history. You can push multiple commits to the remote repository, which for example means that if you are performing changes that would create broken intermediate states, you can commit any number of snapshots locally and only push the whole bundle of commits after you have reached a state that works again.

If another developer has pushed changes since the last time you have pulled from the server, your attempt to push your commits online may fail. In this case you need to pull the remote changes before you can push. Never use force pushing! (`git push -f`)

**Before you commit anything...**

Make sure you have set up your name and email address properly, as described on the Configuring Eclipse page.

### Committing

To commit changes with EGit, do one of the following and select the files you would like to commit:

- Right-click one of the KIELER projects, and from the *Team* submenu, choose *Commit*.
- Right-click the KIELER repository in the *Git Repositories* view and choose *Commit*.

To commit your changes on the command line, do the following:

1. Use `git add` to select (*stage*) the files you would like to commit (new files as well as modified files).
2. Enter `git commit` to create a commit from the staged files.

### Pushing

To push your commits to a remote repository with EGit, do one of the following:

- Right-click one of the KIELER projects, and from the *Team* submenu, choose *Push to Upstream*.
- Right-click the KIELER repository in the *Git Repositories* view and choose *Push to Upstream*.

To push your commits on the command line, enter `git push`.

## Getting Earlier Versions of Files

If you want to revert *all* your local changes and get back to the previous repository state, use `git reset --hard` or the *Reset* item in the EGit context menu. If you only want to revert some specific files, do one of the following:

- Right click the files, click *Replace With*, click *HEAD Revision*.
- Enter `git checkout <path>` on the command line.

By giving a branch, tag, or commit number in `git checkout <commit> <path>`, you can get to any existing version of the files. If no path is given, `git checkout <commit>` switches your whole working copy and index to the specified branch, tag, or commit number. If `git reset` is given a commit number, the current branch is modified to point at the given commit.

> This is a brute force modification, and you probably won't be able to push the new branch

## Cleaning Your Working Directory

In case the working directory is messed up with unstaged files, which are not affected by `git reset --hard`, a clean up is achieved by means of `git clean -f`. The additional switch `-d` applies this to directories, respectively. Hence, `git reset` followed by `git clean` act like `svn revert`.

`git clean -X` removes only the files that are ignored by Git, that is mainly the .class files generated by the Java compiler. A full rebuild is required afterwards.

## Branching and Merging

Branches are used to structure your development and are an essential tool for effective work in the KIELER team. Read the Source Code Management page if you haven't yet understood what branches are good for.

### Creating a Branch

Do one of the following:

- Right-click the KIELER repository in the *Git Repositories* view, click *Switch To*, click *New Branch*.
- On the command line, enter `git branch <name>`. In this case, the new branch is not active; if you want to activate the new branch while creating it, enter `git checkout -b <name>`.

In either case the new branch starts at the current position of your working copy, i.e. it branches from the current branch you are on. In order to branch from a different position, either check out that other branch first or select it as *source ref* in the EGit wizard (on the command line simply type git branch <name> <start_point>).

### Switching to Another Branch

Do one of the following:

- Navigate to the branch in the KIELER repository in the *Git Repositories* view, right click it, click *Checkout*.
- On the command-line, enter `git checkout <name>`.

Note that your local uncommitted changes are transferred when switching the current branch.

### Merging Branches

You always merge another branch into the current branch. Therefore you first have to checkout the target branch prior to merging. Then, do one of the following:

- Navigate to the source branch in the *Git Repositories* view, right-click it, click *Merge.*
- On the command line, enter `git merge <source_name>`.

This creates a new *merge commit*, i.e. a commit with two source commits, and the target branch (the one you're currently on) contains all changes that have been done in the source branch (the one selected for merge). The source branch is not modified. A merge is done implicitly when pulling: assuming you're on branch master, the command git pull origin master is the same as git fetch origin followed by git merge origin/master, where origin/master is the remote tracking branch for master.

## Working With Multiple Remote Repositories

Stash allows the creation of personal server-side clones of the KIELER repository, which is highly encouraged as described on the Source Code Management page. When working with such clones, it is often necessary to synchronize the different server-side repositories with the local one. Git supports this by allowing to configure multiple *remotes* in the local repository. On the command line this is done simply by entering git remote add <name> <url>, where <name> is an arbitrary local identifier for the remote repository. For example, a remote named origin is automatically created when a local repository clone is created through git clone <url>.

When you push or pull branches, simply select the remote you wish to interfere with. Pulling is done by `git pull <remote> <branch>`, and pushing is done by `git push <remote> <branch>`.