

# **kdesrc-build Script Manual**

**Michael Pyne  
Carlos Woelz**



## kdesrc-build Script Manual

# Contents

<b>1</b>	<b>Introduction</b>	<b>8</b>
1.1	A brief introduction to kdesrc-build . . . . .	8
1.1.1	What is kdesrc-build? . . . . .	8
1.1.2	kdesrc-build operation ‘in a nutshell’ . . . . .	8
1.2	Documentation Overview . . . . .	9
<b>2</b>	<b>Getting Started</b>	<b>10</b>
2.1	Preparing the System to Build KDE . . . . .	10
2.1.1	Setup a new user account . . . . .	10
2.1.2	Ensure your system is ready to build KDE software . . . . .	10
2.1.3	Setup kdesrc-build . . . . .	12
2.1.3.1	Install kdesrc-build . . . . .	12
2.1.3.2	Prepare the configuration file . . . . .	12
2.1.3.2.1	Manual setup of configuration file . . . . .	12
2.2	Setting the Configuration Data . . . . .	13
2.3	Using the kdesrc-build script . . . . .	14
2.3.1	Loading project metadata . . . . .	14
2.3.2	Previewing what will happen when kdesrc-build runs . . . . .	14
2.3.3	Resolving build failures . . . . .	15
2.4	Building specific modules . . . . .	16
2.5	Setting the Environment to Run Your KDEPlasma Desktop . . . . .	17
2.5.1	Automatically installing a login driver . . . . .	18
2.5.1.1	Adding xsession support for distributions . . . . .	18
2.5.1.2	Manually adding support for xsession . . . . .	18
2.5.2	Setting up the environment manually . . . . .	19
2.6	Module Organization and selection . . . . .	19
2.6.1	KDE Software Organization . . . . .	19
2.6.2	Selecting modules to build . . . . .	19
2.6.3	Module Sets . . . . .	20
2.6.3.1	The basic module set concept . . . . .	20
2.6.3.2	Special Support for KDE module sets . . . . .	21
2.6.4	The official KDE module database . . . . .	22
2.6.5	Filtering out KDE project modules . . . . .	23
2.7	Getting Started Conclusion . . . . .	23

<b>3</b>	<b>Script Features</b>	<b>25</b>
3.1	Feature Overview . . . . .	25
3.2	kdesrc-build's build logging . . . . .	27
3.2.1	Logging overview . . . . .	27
3.2.1.1	Logging directory layout . . . . .	27
<b>4</b>	<b>Configuring kdesrc-build</b>	<b>28</b>
4.1	Overview of kdesrc-build configuration . . . . .	28
4.1.1	Layout of the configuration file . . . . .	28
4.1.1.1	Global configuration . . . . .	28
4.1.1.2	Module configuration . . . . .	28
4.1.1.3	Processing of option values . . . . .	29
4.1.1.4	'options' modules . . . . .	29
4.1.2	Including other configuration files . . . . .	30
4.1.3	Commonly used configuration options . . . . .	30
4.2	Table of available configuration options . . . . .	31
<b>5</b>	<b>Command Line Options and Environment Variables</b>	<b>65</b>
5.1	Command Line Usage . . . . .	65
5.1.1	Commonly used command line options . . . . .	65
5.1.2	Specifying modules to build . . . . .	66
5.2	Supported Environment Variables . . . . .	66
5.3	Supported command-line parameters . . . . .	66
<b>6</b>	<b>Using kdesrc-build</b>	<b>74</b>
6.1	Preface . . . . .	74
6.2	Basic kdesrc-build features . . . . .	74
6.2.1	qt support . . . . .	74
6.2.2	Standard flags added by kdesrc-build . . . . .	75
6.2.3	Changing kdesrc-build's build priority . . . . .	75
6.2.4	Installation as the superuser . . . . .	76
6.2.5	Showing the progress of a module build . . . . .	76
6.3	Advanced features . . . . .	77
6.3.1	Partially building a module . . . . .	77
6.3.1.1	Checking out portions of a module . . . . .	77
6.3.1.2	Removing directories from a build . . . . .	77
6.3.2	Branching and tagging support for kdesrc-build . . . . .	78
6.3.2.1	What are branches and tags? . . . . .	78
6.3.2.2	How to use branches and tags . . . . .	78
6.3.2.3	Advanced branch support options . . . . .	79
6.3.3	How kdesrc-build tries to ensure a successful build . . . . .	79

## kdesrc-build Script Manual

6.3.3.1	Automatic rebuilds . . . . .	79
6.3.3.2	Manually rebuilding a module . . . . .	80
6.3.4	Changing environment variable settings . . . . .	80
6.3.5	Resuming builds . . . . .	80
6.3.5.1	Resuming a failed or canceled build . . . . .	80
6.3.5.2	Ignoring modules in a build . . . . .	81
6.3.6	Changing options from the command line . . . . .	81
6.3.6.1	Changing global options . . . . .	81
6.3.6.2	Changing module options . . . . .	81
6.4	Features for KDE developers . . . . .	82
6.4.1	SSH Agent checks . . . . .	82
6.5	Other kdesrc-build features . . . . .	82
6.5.1	Changing the amount of output from kdesrc-build . . . . .	82
6.5.2	Color output . . . . .	82
6.5.3	Removing unneeded directories after a build . . . . .	83
<b>7</b>	<b>CMake, the KDE build system</b>	<b>85</b>
7.1	Introduction to CMake . . . . .	85
<b>8</b>	<b>Credits And License</b>	<b>86</b>
<b>A</b>	<b>KDE modules and source code organization</b>	<b>87</b>
A.1	The 'Module' . . . . .	87
A.1.1	Individual modules . . . . .	87
A.1.2	Groups of related modules . . . . .	87
A.1.3	Module 'branch groups' . . . . .	88
<b>B</b>	<b>Superseded profile setup procedures</b>	<b>89</b>
B.1	Setting up a KDE login profile . . . . .	89
B.1.1	Changing your startup profile settings . . . . .	89
B.1.2	Starting KDE . . . . .	90

# List of Tables

4.1	Table of Options . . . . .	64
-----	----------------------------	----

### **Abstract**

kdesrc-build is a script which builds and installs KDE software directly from the KDE project's source code repositories.

# Chapter 1

## Introduction

### 1.1 A brief introduction to kdesrc-build

#### 1.1.1 What is kdesrc-build?

kdesrc-build is a script to help the KDE community install KDE software from its [Git](#) and [Subversion](#) source repositories, and continue to update that software afterwards. It is particularly intended to support those who need to supporting testing and development of KDE software, including users testing bugfixes and developers working on new features.

The kdesrc-build script can be configured to maintain a single individual module, a full Plasma desktop with KDE application set, or somewhere in between.

To get started, see [chapter 2](#), or continue reading for more detail on how kdesrc-build works and what is covered in this documentation.

#### 1.1.2 kdesrc-build operation ‘in a nutshell’

kdesrc-build works by using the tools available to the user at the command-line, using the same interfaces available to the user. When kdesrc-build is run, the following sequence is followed:

1. kdesrc-build reads in the [command line](#) and [configuration file](#), to determine what to build, compile options to use, where to install, etc.
2. kdesrc-build performs a source update for each [module](#). The update continues until all modules have been updated. Modules that fail to update normally do not stop the build – you will be notified at the end which modules did not update.
3. Modules that were successfully updated are built, have their test suite run, and are then installed. To reduce the overall time spent, kdesrc-build will by default start building the code as soon as the first module has completed updating, and allow the remaining updates to continue behind the scenes.

**TIP**

A *very good* overview of how KDE modules are built, including informative diagrams, is provided on [an online article discussing KDE’s Krita application](#). This workflow is what kdesrc-build automates for all KDE modules.



## 1.2 Documentation Overview

This guide is an overview to describe the following aspects of kdesrc-build operation:

- An [overview](#) of the steps required to get started.
- Notable [features](#).
- The [configuration file](#) syntax and options.
- The [command line options](#).

Also documented are the steps which you should perform using other tools (i.e. steps that are not automatically performed by kdesrc-build).

## Chapter 2

# Getting Started

In this chapter, we show how to use the `kdesrc-build` to checkout modules from the KDE repository and build them. We also provide a basic explanation of the KDE source code structure and the steps you have to perform before running the script.

All topics present in this chapter are covered with even more detail in the [Build from Source](#) article, at the [KDE Community Wiki](#). If you are compiling KDE for the first time, it is a good idea to read it, or consult it as a reference source. You will find detailed information about packaging tools and requirements, common compilation pitfalls and strategies and information about running your new KDE installation.

## 2.1 Preparing the System to Build KDE

### 2.1.1 Setup a new user account

It is recommended that you use a different user account to build, install, and run your KDE software from, since less permissions are required, and to avoid interfering with your distribution's packages. If you already have KDE packages installed, the best choice would be to create a different (dedicated) user to build and run the new KDE.

**TIP**

Leaving your system KDE untouched also allows you to have an emergency fallback in case a coding mistake causes your latest software build to be unusable.

You can do also setup to install to a system-wide directory (e.g. `/usr/src/local`) if you wish. This document does not cover this installation type, since we assume you know what you are doing.

### 2.1.2 Ensure your system is ready to build KDE software

Before using the `kdesrc-build` script (or any other building strategy) you must install the development tools and libraries needed for KDE. The nearly complete list of required tools can be found from the [KDE Community Wiki Build Requirements](#) page.

Here is a list of some of the things you will need:

## kdesrc-build Script Manual

- You will need CMake, this software is what KDE uses to handle build-time configuration of the source code and generation of the specific build commands for your system. The required version will vary depending on what versions of KDE software you are building (see Tech-Base for specifics), but with modern distributions the CMake included with your distribution should be quite sufficient.
- You must also install the source control clients needed to checkout the KDE source code. This means you need at least the following:
  - The [Git source control manager](#), which is used for all KDE [source code](#)
  - Although it is not required, the [Bazaar](#) source control manager is used for a single module (libdbusmenu-qt) that is required for the KDE libraries. Most users can install this library through their distribution packages but kdesrc-build supports building it as well if you desire. But to build libdbusmenu-qt, you must have Bazaar installed.
- The Perl scripting language is required for kdesrc-build, some KDE repositories, and Qt™ (if you build that from source).

The Perl that comes with your distribution should be suitable (it needs to be at least Perl 5.14), but you will also need some additional modules (kdesrc-build will warn if they are not present):

- IO::Socket::SSL
  - JSON::PP or JSON::XS
  - YAML::PP, YAML::XS, or YAML::Syck
- You will need a full C++ development environment (compiler, standard library, runtime, and any required development packages). The minimum required versions vary based on the KDE module: the KDE Frameworks 5 collection supports the oldest compilers, while KDE Plasma 5 and KDE Applications tend to require more recent compilers.

The GCC 4.8 or Clang 4 compilers are the minimum recommended. Many distributions support easily installing these tools using a ‘build-essentials’ package, an option to install “build dependencies” with Qt™, or similar features. The KDE Community Wiki has a page [tracking recommended packages for major distributions](#).
  - You will need a build tool that actually performs the compilation steps (as generated by CMake). GNU Make is recommended and should be available through your package manager. CMake does support others options, such as the Ninja build tool, which can be used by kdesrc-build using the [custom-build-command](#) configuration file option.
  - Finally, you will need the appropriate Qt™ libraries (including development packages) for the version of KDE software you are building. kdesrc-build does not officially support building Qt™ 5 (the current major version), so it is recommended to use your distribution’s development packages or to see the KDE Community wiki page on [self-building Qt 5](#).

### NOTE

Most operating system distributions include a method of easily installing required development tools. Consult the Community Wiki page [Required devel packages](#) to see if these instructions are already available.

### IMPORTANT

Some of these packages are divided into libraries (or programs or utilities), and development packages. You will need at least the program or library *and* its development package.

## 2.1.3 Setup kdesrc-build

### 2.1.3.1 Install kdesrc-build

The KDE developers make frequent changes to kdesrc-build to keep it in sync with advances in KDE development, including improvements to the recommended kdesrc-build configuration, added modules, improving CMake flags, etc.

Because of this, we recommend obtaining kdesrc-build directly from its source repository and then periodically updating it.

You can obtain kdesrc-build from its source repository by running:

```
$ git clone https://invent.kde.org/sdk/kdesrc-build.git ~/kdesrc-build
```

Replace `~/kdesrc-build` with the directory you would like to install to.

You can update kdesrc-build later by running:

```
$ cd ~/kdesrc-build
$ git pull
```

#### TIP

We recommend adding the kdesrc-build installation directory to your `PATH` environment variable, so that you can run kdesrc-build without having to fully specify its path every time.

### 2.1.3.2 Prepare the configuration file

kdesrc-build uses a [configuration file](#) (located at `~/kdesrc-buildrc`) to control which modules are built, where they are installed to, etc.

You can use a program included with kdesrc-build, called `kdesrc-build-setup` in order to prepare a simple kdesrc-build configuration. You can then edit the `~/kdesrc-buildrc` from there to make any changes you see fit.

`kdesrc-build-setup` itself runs from a terminal (instead of using a graphical interface), just like kdesrc-build, so you can use it even if you have no graphical interface available yet.

#### 2.1.3.2.1 Manual setup of configuration file

You can also setup your configuration file manually, by copying the included sample configuration file `kdesrc-buildrc-kf5-sample` to `~/kdesrc-buildrc` and then editing the file. [chapter 4](#) will be a useful reference for this, especially its [table of configuration options](#).

kdesrc-build contains many recommended configuration files to support KDE Frameworks 5, Plasma 5, and other KDE applications. The `kdesrc-build-setup` refers to these files in the configuration file it generates, but you can also use them yourself. See [Section 4.1.2](#) for information on how to use other configuration files from your own `~/kdesrc-buildrc`.

You can find more information about the syntax of the [configuration file](#) in [Section 2.2](#) and in [chapter 4](#).

## 2.2 Setting the Configuration Data

To use `kdesrc-build`, you should have a file in your home directory called `.kdesrc-buildrc`, which sets the general options and sets the modules you would like to download and build.

### NOTE

It is possible to use different configuration files for `kdesrc-build`, which is described in chapter 4. If you need to use multiple configurations, please see that section. Here, we will assume the configuration is stored in `~/.kdesrc-buildrc`.

The easiest way to proceed is to use the `kdesrc-buildrc-kf5-sample` file as a template, changing global options to match your wants, and also change the list of modules you want to build.

The default settings should be appropriate to perform a KDE build. Some settings that you may wish to alter include:

- `kdedir`, which changes the destination directory that your KDE software is installed to. This defaults to `~/kde`, which is a single-user installation.
- `branch-group`, which can be used to choose the appropriate branch of development for the KDE modules as a whole. There are many supported build configurations but you will likely want to choose `kf5-qt5` so that `kdesrc-build` downloads the latest code based on Qt™ 5 and KDE Frameworks 5.

### TIP

`kdesrc-build` will use a default branch group if you do not choose one, but this default will change over time, so it's better to choose one so that the branch group does not change unexpectedly.

- `source-dir`, to control the directory `kdesrc-build` uses for downloading the source code, running the build process, and saving logs. This defaults to `~/kdesrc`.
- `cmake-options`, which sets the options to pass to the CMake command when building each module. Typically this is used to set between 'debug' or 'release' builds, to enable (or disable) optional features, or to pass information to the build process about the location of required libraries.
- `make-options`, which sets the options used when actually running the make command to build each module (once CMake has established the build system).

The most typical option is `-jN`, where `N` should be replaced with the maximum number of compile jobs you wish to allow. A higher number (up to the number of logical CPUs your system has available) leads to quicker builds, but requires more system resources.

### TIP

`kdesrc-build` sets the option `num-cores` to the detected number of available processing cores. You can use this value in your own configuration file to avoid having to set it manually.

**Example 2.1** Configuring Make to use all available CPUs, with exceptions

```

global
  # This environment variable is automatically used by make, including
  # make commands not run by kdesrc-build directly, such as Qt's ↔
  configure
  set -env MAKEFLAGS -j${num-cores}
  &#8230;
end global

&#8230;

module-set big-module-set
  repository kde-projects
  use-modules calligra
  make-options -j2 # Reduced number of build jobs for just these modules
end module-set

```

**NOTE**

Some very large Git repositories may swamp your system if you try to compile with a too many build jobs at one time, especially repositories like the Qt™ WebKit and Qt™ WebEngine repositories. To maintain system interactivity you may have to reduce the number of build jobs for specific modules. Example 2.1 gives an example of how to do this.

You may want to select different modules to build, which is described in Section 2.6.2.

## 2.3 Using the kdesrc-build script

With the configuration data established, now you are ready to run the script. Even if you still have some tweaking or other reading you wish to do, it is a good idea to at least load the KDE project metadata.

### 2.3.1 Loading project metadata

From a terminal window, log in to the user you are using to compile KDE software and execute the script:

```
% kdesrc-build --metadata-only
```

This command will setup the source directory and connect to the KDE Git repositories to download the database of KDE git repositories, and the database of dependency metadata, without making any further changes. It is useful to run this separately as this metadata is useful for other kdesrc-build commands.

### 2.3.2 Previewing what will happen when kdesrc-build runs

With the project metadata installed, it is possible to preview what kdesrc-build will do when launched. This can be done with the `--pretend` command line option.

```
% ./kdesrc-build --pretend
```

## kdesrc-build Script Manual

You should see a message saying that some packages were successfully built (although nothing was actually built). If there were no significant problems shown, you can proceed to actually running the script.

```
% kdesrc-build --stop-on-failure
```

This command will download the appropriate source code, and build and install each module in order, but will stop if a module fails to build (due to the `--stop-on-failure` option). Afterwards, you should see output similar to that in [Example 2.2](#):

---

### Example 2.2 Example output of a kdesrc-build run

---

```
% kdesrc-build
Updating kde-build-metadata (to branch master)
Updating sysadmin-repo-metadata (to branch master)

Building libdbusmenu-qt (1/200)
  No changes to libdbusmenu-qt source, proceeding to build.
  Compiling... succeeded (after 0 seconds)
  Installing.. succeeded (after 0 seconds)

Building taglib (2/200)
  Updating taglib (to branch master)
  Source update complete for taglib: 68 files affected.
  Compiling... succeeded (after 0 seconds)
  Installing.. succeeded (after 0 seconds)

Building extra-cmake-modules from <module-set at line 32> (3/200)
  Updating extra-cmake-modules (to branch master)
  Source update complete for extra-cmake-modules: 2 files affected.
  Compiling... succeeded (after 0 seconds)
  Installing.. succeeded (after 0 seconds)

  ...

Building kdevelop from kdev (200/200)
  Updating kdevelop (to branch master)
  Source update complete for kdevelop: 29 files affected.
  Compiling... succeeded (after 1 minute, and 34 seconds)
  Installing.. succeeded (after 2 seconds)

<<<  PACKAGES SUCCESSFULLY BUILT  >>>
Built 200 modules

Your logs are saved in /home/kde-src/kdesrc/log/2018-01-20-07
```

### 2.3.3 Resolving build failures

Depending on how many modules you are downloading, it is possible that `kdesrc-build` will not succeed the first time you compile KDE software. Do not despair!

`kdesrc-build` logs the output of every command it runs. By default, the log files are kept in `~/kde/src/log`. To see what caused an error for a module in the last `kdesrc-build` command, usually it is sufficient to look at `~/kdesrc/log/latest/ module-name /error.log`.

**TIP**

Perhaps the easiest way to find out what error caused a module to fail to build is to search backward with a case-insensitive search, starting from the end of the file looking for the word `error`. Once that is found, scroll up to make sure there are no other error messages nearby. The first error message in a group is usually the underlying problem.

In that file, you will see the error that caused the build to fail for that module. If the file says (at the bottom) that you are missing some packages, try installing the package (including any appropriate `-dev` packages) before trying to build that module again. Make sure that when you run `kdesrc-build` again to pass the `--reconfigure` option so that `kdesrc-build` forces the module to check for the missing packages again.

Or, if the error appears to be a build error (such as a syntax error, ‘incorrect prototype’, ‘unknown type’, or similar) then it is probably an error with the KDE source, which will hopefully be resolved within a few days. If it is not resolved within that time, feel free to mail the [kde-devel@kde.org](mailto:kde-devel@kde.org) mailing list (subscription may be required first) in order to report the build failure.

You can find more common examples of things that can go wrong and their solutions, as well as general tips and strategies to build KDE software in the [Build from Source](#).

On the other hand, assuming everything went well, you should have a new KDE install on your computer, and now it is simply a matter of running it, described next in [Section 2.5](#).

**NOTE**

For more information about `kdesrc-build`’s logging features, please see [Section 3.2](#).

## 2.4 Building specific modules

Rather than building every module all the time, you may only want to build a single module, or other small subset. Rather than editing your configuration file, you can simply pass the names of modules or module sets to build to the command line.



## kdesrc-build Script Manual

### Example 2.3 Example output of a kdesrc-build specific module build

```
% kdesrc-build --include-dependencies dolphin
Updating kde-build-metadata (to branch master)
Updating sysadmin-repo-metadata (to branch master)

Building extra-cmake-modules from frameworks-set (1/79)
  Updating extra-cmake-modules (to branch master)
  No changes to extra-cmake-modules source, proceeding to build.
  Running cmake...
  Compiling... succeeded (after 0 seconds)
  Installing.. succeeded (after 0 seconds)

Building phonon from phonon (2/79)
  Updating phonon (to branch master)
  No changes to phonon source, proceeding to build.
  Compiling... succeeded (after 0 seconds)
  Installing.. succeeded (after 0 seconds)

Building attica from frameworks-set (3/79)
  Updating attica (to branch master)
  No changes to attica source, proceeding to build.
  Compiling... succeeded (after 0 seconds)
  Installing.. succeeded (after 0 seconds)

  ...

Building dolphin from base-apps (79/79)
  Updating dolphin (to branch master)
  No changes to dolphin source, proceeding to build.
  Compiling... succeeded (after 0 seconds)
  Installing.. succeeded (after 0 seconds)

<<<  PACKAGES SUCCESSFULLY BUILT  >>>
Built 79 modules

Your logs are saved in /home/kde-src/kdesrc/log/2018-01-20-07
```

In this case, although only the *dolphin* application was specified, the `--include-dependencies` flag caused `kdesrc-build` to include the dependencies listed for *dolphin* (by setting the `include-dependencies` option).

#### NOTE

The dependency resolution worked in this case only because *dolphin* happened to be specified in a kde-projects-based module set (in this example, named `base-apps`). See Section 2.6.3.2.

## 2.5 Setting the Environment to Run Your KDEPlasma Desktop

Assuming you are using a dedicated user to build KDE Plasma, and you already have an installed Plasma version, running your new Plasma may be a bit tricky, as the new Plasma has to take precedence over the old. You must change the environment variables of your login scripts to make sure the newly-built desktop is used.

## 2.5.1 Automatically installing a login driver

Starting from version 1.16, kdesrc-build will try to install an appropriate login driver, that will allow you to login to your kdesrc-build-built KDE desktop from your login manager. This can be disabled by using the `install-session-driver` configuration file option.

### NOTE

Session setup does not occur while kdesrc-build is running in pretend mode.

This driver works by setting up a custom ‘xsession’ session type. This type of session should work by default with the kdm login manager (where it appears as a ‘Custom’ session), but other login managers (such as LightDM and gdm) may require additional files installed to enable xsession support.

### 2.5.1.1 Adding xsession support for distributions

The default login managers for some distributions may require additional packages to be installed in order to support xsession logins.

- The [Fedora Linux](#)<sup>®</sup> distribution requires the `xorg-x11-xinit-session` package to be installed for custom xsession login support.
- [Debian](#) and Debian-derived Linux<sup>®</sup> distributions should support custom xsession logins, but require the `allow-user-xsession` option to be set in `/etc/X11/Xsession.options`. See also the [Debian documentation on customizing the X session](#).
- For other distributions, go to [Section 2.5.1.2](#).

### 2.5.1.2 Manually adding support for xsession

If there were no distribution-specific directions for your distribution in [Section 2.5.1.1](#), you can manually add a ‘Custom xsession login’ entry to your distribution’s list of session types as follows:

### NOTE

This procedure will likely require administrative privileges to complete.

1. Create the file `/usr/share/xsessions/kdesrc-build.desktop`.
2. Ensure the file just created has the following text:

```
Type=XSession
Exec=$HOME/.xsession
Name=KDE Plasma Desktop (unstable; kdesrc-build)
```

- The `$HOME` entry must be replaced by the full path to your home directory (example, `/home/ user`). The desktop entry specification does not allow for user-generic files.
3. When the login manager is restarted, it should show a new session type, ‘KDE Plasma Desktop (unstable; kdesrc-build)’ in its list of sessions, which should try to run the `.xsession` file installed by kdesrc-build if it is selected when you login.

### NOTE

It may be easiest to restart the computer to restart the login manager, if the login manager does not track updates to the `/usr/share/xsessions` directory.

## 2.5.2 Setting up the environment manually

This documentation used to include instruction on which environment variables to set in order to load up the newly-built desktop. These instructions have been moved to an appendix (Section B.1).

If you intend to setup your own login support you can consult that appendix or view the `sample-kde-env-master.sh` file included with the `kdesrc-build` source.

## 2.6 Module Organization and selection

### 2.6.1 KDE Software Organization

KDE software is split into different components, many of which can be built by `kdesrc-build`. Understanding this organization will help you properly select the software modules that you want built.

1. At the lowest level comes the Qt™ library, which is a very powerful, cross-platform ‘toolkit’ library. KDE is based on Qt™, and some of the non-KDE libraries required by KDE are also based on Qt™. `kdesrc-build` can build Qt™, or use the one already installed on your system if it is a recent enough version.
2. On top of Qt™ are required libraries that are necessary for KDE software to work. Some of these libraries are not considered part of KDE itself due to their generic nature, but are still essential to the KDE Platform. These libraries are collected under a `kdesupport` module grouping but are not considered part of the ‘Frameworks’ libraries.
3. On top of these essential libraries come the [KDE Frameworks](#), sometimes abbreviated as KF5, which are essential libraries for the KDE Plasma desktop, KDE Applications, and other third-party software.
4. On top of the Frameworks, come several different things:
  - ‘Third-party’ applications. These are applications that use the KDE Frameworks or are designed to run under KDE Plasma but are not authored by or in association with the KDE project.
  - Plasma, which is a full ‘workspace’ desktop environment. This is what users normally see when they ‘log-in to KDE’.
  - The KDE Application suite. This is a collection of useful software included with the Platform and Plasma Desktop, grouped into individual modules, including utilities like Dolphin, games like KSudoku, and productivity software released by KDE such as Kontact.
  - Finally, there is a collection of software (also collected in modules) whose development is supported by KDE resources (such as translation, source control, bug tracking, etc.) but is not released by KDE as part of Plasma or the Application suite. These modules are known as ‘Extragear’.

### 2.6.2 Selecting modules to build

Selecting which of the possible modules to build is controlled by [the configuration file](#). After the `global` section is a list of modules to build, bracketed by `module ... end module` lines. An example entry for a module is shown in [Example 2.4](#).

---

### Example 2.4 Example module entry in the configuration file

---

```
module kdesrc-build-git
  # Options for this module go here, example:
  repository kde:kdesrc-build
  make-options -j4 # Run 4 compiles at a time
end module
```

---

#### NOTE

In practice, this module construct is not usually used directly. Instead most modules are specified via module-sets as described below.

When using only `module` entries, `kdesrc-build` builds them in the order you list, and does not attempt to download any other repositories other than what you specify directly.

## 2.6.3 Module Sets

The KDE source code is decomposed into a great number of relatively small Git-based repositories. To make it easier to manage the large number of repositories involved in any useful KDE-based install, `kdesrc-build` supports grouping multiple modules and treating the group as a ‘module set’.

### 2.6.3.1 The basic module set concept

By using a module set, you can quickly declare many Git modules to be downloaded and built, as if you’d typed out a separate module declaration for each one. The `repository` option is handled specially to setup where each module is downloaded from, and every other option contained in the module set is copied to every module generated in this fashion.

---

### Example 2.5 Using module sets

---

```
global
  git-repository-base kde-git kde:
end global

module qt
  # Options removed for brevity
end module

module-set kde-support-libs
  repository kde-git
  use-modules automoc attica akonadi
end module-set

# Other modules as necessary...
module kdesupport
end module
```

---

In Example 2.5 a brief module set is shown. When `kdesrc-build` encounters this module set, it acts as if, for every module given in `use-modules`, that an individual module has been declared, with

its `repository` equal to the module-set's `repository` followed immediately by the given module name.

In addition, other options can be passed in a module set, which are copied to every new module that is created this way. By using `module-set` it is possible to quickly declare many Git modules that are all based on the same repository URL. In addition, it is possible to give module-sets a name (as shown in the example), which allows you to quickly refer to the entire group of modules from the command line.

### 2.6.3.2 Special Support for KDE module sets

The module set support described so far is general to any Git-based modules. For the KDE Git repositories, `kdesrc-build` includes additional features to make things easier for users and developers. This support is enabled by specifying `kde-projects` as the `repository` for the module set.

`kdesrc-build` normally only builds the modules you have listed in your configuration file, in the order you list them. But with a `kde-projects` module set, `kdesrc-build` can do dependency resolution of KDE-specific modules, and in addition automatically include modules into the build even if only indirectly specified.

---

#### Example 2.6 Using `kde-projects` module sets

---

```
# Only adds a module for juk (the kde/kdemultimedia/juk repo)
module-set juk-set
  repository kde-projects
  use-modules juk
end module-set

# Adds all modules that are in kde/multimedia/*, including juk,
# but no other dependencies
module-set multimedia-set
  repository kde-projects
  use-modules kde/multimedia
end module-set

# Adds all modules that are in kde/multimedia/*, and all kde-projects
# dependencies from outside of kde/kdemultimedia
module-set multimedia-deps-set
  repository kde-projects
  use-modules kde/multimedia
  include-dependencies true
end module-set

# All modules created out of these three module sets are automatically put ←
# in
# proper dependency order, regardless of the setting for include- ←
# dependencies
```

---

#### TIP

This `kde-projects` module set construct is the main method of declaring which modules you want to build.

All module sets use the `repository` and `use-modules` options. `kde-projects` module sets have a predefined `repository` value, but other types of module sets also will use the `git-repository-base` option.

## 2.6.4 The official KDE module database

KDE's Git repositories allow for grouping related Git modules into collections of related modules (e.g. kdegraphics). Git doesn't recognize these groupings, but kdesrc-build can understand these groups, using [module sets](#) with a `repository` option set to 'kde-projects'.

kdesrc-build will recognize that the `kde-projects` repository requires special handling, and adjust the build process appropriately. Among other things, kdesrc-build will:

- Download the latest module database from the [KDE git archive](#).
- Try to find a module with the name given in the module set's `use-modules` setting in that database.
- For every module that is found, kdesrc-build will lookup the appropriate repository in the database, based upon the [branch-group](#) setting in effect. If a repository exists and is active for the branch group, kdesrc-build will automatically use that to download or update the source code.

### NOTE

In the current database, some module groups not only have a collection of modules, but they *also* declare their own Git repository. In these situations kdesrc-build will currently prefer the group's Git repository instead of including the childrens' repositories.

The following example shows how to use the KDE module database to install the Phonon multimedia library.

```
module-set media-support
  # This option must be kde-projects to use the module database.
  repository kde-projects

  # This option chooses what modules to look for in the database.
  use-modules phonon/phonon phonon-gstreamer phonon-vlc
end module-set
```

### TIP

`phonon/phonon` is used since (with the current project database) kdesrc-build would otherwise have to decide between the group of projects called 'phonon' or the individual project named 'phonon'. Currently kdesrc-build would pick the former, which would build many more backends than needed.

The following example is perhaps more realistic, and shows a feature only available with the KDE module database: Building all of the KDE graphics applications with only a single declaration.

```
module-set kdegraphics
  # This option must be kde-projects to use the module database.
  repository kde-projects

  # This option chooses what modules to look for in the database.
  use-modules kdegraphics/libs kdegraphics/*
end module-set
```

There are two important abilities demonstrated here:

1. kdesrc-build allows you to specify modules that are descendants of a given module, without building the parent module, by using the syntax `module-name /*`. It is actually required in this case since the base module, `kdegraphics`, is marked as inactive so that it is not accidentally built along with its children modules. Specifying the descendent modules allows kdesrc-build to skip around the disabled module.
2. kdesrc-build will also not add a given module to the build list more than once. This allows us to manually set `kdegraphics/libs` to build first, before the rest of `kdegraphics`, without trying to build `kdegraphics/libs` twice. This used to be required for proper dependency handling, and today remains a fallback option in case the KDE project database is missing dependency metadata.

### 2.6.5 Filtering out KDE project modules

You might decide that you'd like to build all programs within a KDE module grouping *except* for a given program.

For instance, the `kdeutils` group includes a program named `kremotecontrol`. If your computer does not have the proper hardware to receive the signals sent by remote controls then you may decide that you'd rather not download, build, and install `kremotecontrol` every time you update `kdeutils`.

You can achieve this by using the `ignore-modules` configuration option. On the command line the `--ignore-modules` option does the same thing, but is more convenient for filtering out a module just once.

---

#### Example 2.7 Example for ignoring a kde-project module in a group

---

```
module-set utils
  repository kde-projects

  # This option chooses what modules to look for in the database.
  use-modules kdeutils

  # This option "subtracts out" modules from the modules chosen by use- ←
  modules, above.
  ignore-modules kremotecontrol
end module-set

module-set graphics
  repository kde-projects

  # This option chooses what modules to look for in the database.
  use-modules extragear/graphics

  # This option "subtracts out" modules from the modules chosen by use- ←
  modules, above.
  # In this case, *both* extragear/graphics/kipi-plugins and
  # extragear/graphics/kipi-plugins/kipi-plugins-docs are ignored
  ignore-modules extragear/graphics/kipi-plugins
end module-set
```

---

## 2.7 Getting Started Conclusion

These are the major features and concepts needed to get started with kdesrc-build

## kdesrc-build Script Manual

For additional information, you could keep reading through this documentation. In particular, the [list of command-line options](#) and the [table of configuration file options](#) are useful references.

The KDE Community also maintains [an online Wiki reference for how to build the source code](#), which refers to kdesrc-build and includes tips and other guidelines on how to use the tool.



## Chapter 3

# Script Features

### 3.1 Feature Overview

kdesrc-build features include:

- You can ‘pretend’ to do the operations. If you pass `--pretend` or `-p` on the command line, the script will give a verbose description of the commands it is about to execute, without actually executing it. However if you’ve never run `kdesrc-build`, you would want to run the `kdesrc-build --metadata-only` command first in order for `--pretend` to work.

**TIP**

For an even more verbose description of what `kdesrc-build` is doing, try using the `--debug` option.

- `kdesrc-build` can (with the assistance of the KDE FTP server) allow for speedy checkouts of some Subversion modules. If the module you are checking out has already been packaged at the website, then `kdesrc-build` will download the snapshot and prepare it for use on your computer.

**TIP**

There is generally no need for any special preparation to perform the initial checkout of a Git module, as the entire Git repository must be downloaded anyways, so it is easy for the server to determine what to send.

This is faster for you, and helps to ease the load on the `kde.org` anonymous Subversion servers.

- Another speedup is provided by starting the build process for a module as soon as the source code for that module has been downloaded. (Available since version 1.6)
- Excellent support for building the Qt™ library (in case the KDE software you are trying to build depends on a recent Qt™ not available in your distribution).
- `kdesrc-build` does not require a GUI present to operate. So, you can build KDE software without needing a graphical environment.
- Supports setting default options for all modules (such as the compilation settings or the configuration options). Such options can normally be changed for specific modules as well.

Also, `kdesrc-build` will [add standard flags](#) as appropriate to save you the trouble and possible errors from typing them yourself. Nota Bene: this does not apply when a (custom) toolchain is configured through e.g.: [cmake-toolchain](#)

## kdesrc-build Script Manual

- kdesrc-build can checkout a specific [branch or tag](#) of a module. You can also ensure that a specific [revision](#) is checked out of a module.
- kdesrc-build can automatically switch a source directory to checkout from a different repository, branch, or tag. This happens automatically when you change an option that changes what the repository URL should be, but you must use the `--src-only` option to let kdesrc-build know that it is acceptable to perform the switch.
- kdesrc-build can [checkout only portions of a module](#), for those situations where you only need one program from a large module.
- For developers: kdesrc-build will [remind you](#) if you use `svn+ssh://` but `ssh-agent` is not running, as this will lead to repeated password requests from SSH.
- Can [delete the build directory](#) of a module after its installation to save space at the expense of future compilation time.
- The locations for the directories used by kdesrc-build are configurable (even per module).
- Can use Sudo, or a different user-specified command to [install modules](#) so that kdesrc-build does not need to be run as the super user.
- kdesrc-build runs [with reduced priority](#) by default to allow you to still use your computer while kdesrc-build is working.
- Has support for using KDE's [tags and branches](#).
- There is support for [resuming a build](#) from a given module. You can even [ignore some modules](#) temporarily for a given build.
- kdesrc-build will show the [progress of your build](#) when using CMake, and will always time the build process so you know after the fact how long it took.
- Comes built-in with a sane set of default options appropriate for building a base KDE single-user installation from the anonymous source repositories.
- Tilde-expansion for your configuration options. For example, you can specify:

```
qtdir ~/kdesrc/build/qt
```

- Automatically sets up a build system, with the source directory not the same as the build directory, in order to keep the source directory pristine.
- You can specify global options to apply to every module to check out, and you can specify options to apply to individual modules as well.
- Forced full rebuilds, by running kdesrc-build with the `--refresh-build` option.
- You can specify various environment values to be used during the build, including `KDEDIR`, `QTDIR`, `DO_NOT_COMPILE`, and `CXXFLAGS`.
- Command logging. Logs are dated and numbered so that you always have a log of a script run. Also, a special symlink called `latest` is created to always point to the most recent log entry in the log directory.
- You can check out only a portion of a KDE Subversion module. For example, you could check out only the `taglib` from `kdesupport`.

## 3.2 kdesrc-build's build logging

### 3.2.1 Logging overview

Logging is a kdesrc-build feature whereby the output from every command that kdesrc-build runs is saved to a file for examination later, if necessary. This is done because it is often necessary to have the output of these programs when there is a build failure, because there are so many reasons why a build can fail in the first place.

#### 3.2.1.1 Logging directory layout

The logs are always stored under the log directory. The destination of the log directory is controlled by the `log-dir` option, which defaults to `${source-dir} /log` (where `${source-dir}` is the value of the `source-dir` option. The in rest of this section, this value will be referred to as `${log-dir}`).

Under `${log-dir}`, is a set of directories, one for every time that kdesrc-build was run. Each directory is named with the date, and the run number. For instance, the second time that kdesrc-build is run on May 26, 2004, it would create a directory called `2004-05-26-02`, where the `2004-05-26` is for the date, and the `-02` is the run number.

For your convenience, kdesrc-build will also create a link to the logs for your latest run, called `latest`. So the logs for the most recent kdesrc-build run should always be under `${log-dir} /latest`.

Now, each directory for a kdesrc-build run will itself contain a set of directories, one for every KDE module that kdesrc-build tries to build. Also, a file called `build-status` will be contained in the directory, which will allow you to determine which modules built and which failed.

#### NOTE

If a module itself has a submodule (such as `extragear/multimedia`, `playground/utils`, or `KDE/kdelibs`), then there would actually be a matching layout in the log directory. For example, the logs for `KDE/kdelibs` after the last kdesrc-build run would be found in `${log-dir} /latest/KDE/kdelibs`, and not under `${log-dir} /latest/kdelibs`.

In each module log directory, you will find a set of files for each operation that kdesrc-build performs. If kdesrc-build updates a module, you may see filenames such as `svn-co.log` (for a module checkout) or `svn-up.log` (when updating a module that has already been checked out). If the `configure` command was run, then you would expect to see a `configure.log` in that directory.

If an error occurred, you should be able to see an explanation of why in one of the files. To help you determine which file contains the error, kdesrc-build will create a link from the file containing the error (such as `build-1.log` to a file called `error.log`).

The upshot to all of this is that to see why a module failed to build after your last kdesrc-build, the file you should look at first is `${log-dir} /latest/ module-name /error.log`.

#### TIP

If the file `error.log` is empty (especially after an installation), then perhaps there was no error. Some of the tools used by the KDE build system will sometimes mistakenly report an error when there was none.

Also, some commands will evade kdesrc-build's output redirection and bypass the log file in certain circumstances (normally when performing the first Subversion checkout), and the error output in that case is not in the log file but is instead at the Konsole or terminal where you ran kdesrc-build.

## Chapter 4

# Configuring kdesrc-build

### 4.1 Overview of kdesrc-build configuration

To use the script, you must have a file in your home directory called `.kdesrc-buildrc`, which describes the modules you would like to download and build, and any options or configuration parameters to use for these modules.

#### 4.1.1 Layout of the configuration file

##### 4.1.1.1 Global configuration

The configuration file starts with the global options, specified like the following:

```
global
option-name option-value
[...]
end global
```

##### 4.1.1.2 Module configuration

It is then followed by one or more module sections, specified in one of the following two forms:

- ```
module module-name
option-name option-value
[...]
end module
```

- ```
module-set module-set-name
  repository kde-projects or git://host.org/path/to/repo.git
  use-modules module-names

# Other options may also be set
option-name option-value
[...]
end module-set
```

**IMPORTANT**

Note that the second form, `module sets`, *only works for Git-based modules*.

For Subversion modules, `module-name` must be a module from the KDE Subversion repository (for example, `kdeartwork` or `kde-wallpapers`), although it is possible to get around this if you [manually specify the Subversion URL](#).

For Git modules, the module name can be essentially whatever you'd like, as long as it does not duplicate any other module name in the configuration. Keep in mind the source and build directory layout will be based on the module name if you do not use the `dest-dir` option.

However, for Git *module sets* the `module-names` must correspond with actual git modules in the chosen repository. See [git-repository-base](#) or [use-modules](#) for more information.

#### 4.1.1.3 Processing of option values

In general, the entire line contents after the `option-name` is used as the `option-value`.

One modification that `kdesrc-build` performs is that a sequence `#{name-of-option}` is replaced with the value of that option from the global configuration. This allows you to reference the value of existing options, including options already set by `kdesrc-build`.

To see an example of this in use, see [Example 2.1](#).

#### 4.1.1.4 'options' modules

There is a final type of configuration file entry, `options` groups, which may be given wherever a `module` or `module-set` may be used.

```
options module-name
option-name option-value
[...]
end options
```

An `options` group may have options set for it just like a module declaration, and is associated with an existing module. Any options set these way will be used to *override* options set for the associated module.

**IMPORTANT**

The associated module name *must* match the name given in the `options` declaration. Be careful of mis-typing the name.

This is useful to allow for declaring an entire `module-set` worth of modules, all using the same options, and then using `options` groups to make individual changes.

`options` groups can also apply to named module sets. This allows expert users to use a common configuration file (which includes `module-set` declarations) as a baseline, and then make changes to the options used by those module-sets in configuration files that use the `include` command to reference the base configuration.

**Example 4.1** Example of using options

In this example we choose to build all modules from the KDE multimedia software grouping. However we want to use a different version of the KMix application (perhaps for testing a bug fix). It works as follows:

```
module-set kde-multimedia-set
  repository kde-projects
  use-modules kde/kdemultimedia
  branch master
end module-set

# kmix is a part of kde/kdemultimedia group, even though we never named
# kmix earlier in this file, kdesrc-build will figure out the change.
options kmix
  branch KDE/4.12
end options
```

Now when you run `kdesrc-build`, all of the KDE multimedia programs will be built from the 'master' branch of the source repository, but KMix will be built from the older 'KDE/4.12' branch. By using `options` you didn't have to individually list all the *other* KDE multimedia programs to give them the right branch option.

**NOTE**

Note that this feature is only available in `kdesrc-build` from version 1.16, or using the development version of `kdesrc-build` after 2014-01-12.

**4.1.2 Including other configuration files**

Within the configuration file, you may reference other files by using the `include` keyword with a file, which will act as if the file referenced had been inserted into the configuration file at that point.

For example, you could have something like this:

```
global
  include ~/common-kdesrc-build-options

  # Insert specific options here.
end global
```

**NOTE**

If you don't specify the full path to the file to include, then the file will be searched for starting from the directory containing the source file. This works recursively as well.

**4.1.3 Commonly used configuration options**

The following is a list of commonly-used options. Click on the option to find out more about it. To see the full list of options, see Section 4.2.

- [cmake-options](#) to define what flags to configure a module with using CMake.

- [branch](#), to checkout from a branch instead of /trunk (for Subversion) or master (for Git).
- [configure-flags](#) to define what flags to configure Qt™ with.
- [kddir](#), to set the directory to install KDE to.
- [make-options](#), to pass options to the Make program (such as number of CPUs to use).
- [qtdir](#), to set the path to Qt™.
- [source-dir](#), to change where to download the source code to.

## 4.2 Table of available configuration options

Here is a table of the various options, containing the following information:

- The option name
- A description of how kdesrc-build responds if the option is set in both the global section, and the module section of the [configuration file](#) while building a module.
- Special comments on the purpose and usage of the option.

Option-name	Module -> Global Behavior	Notes
apidox		This option was used to allow for building KDE module API documentation. It was removed in kdesrc-build 1.6.3 due to lack of support. Online API documentation is available from <a href="http://kde.org">kde.org</a> . In addition it is possible to build KDE API documentation using the <b>kdedoxgen.sh</b> script included in the kde-dev-scripts module. See <a href="#">KDE TechBase</a> for more details.
apply-qt-patches		This option was removed in kdesrc-build 1.10. To get the same effect, see <a href="#">Section 6.2.1</a> and the <a href="#">repository</a> option.

## kdesrc-build Script Manual

<p>async</p>	<p>Cannot be overridden</p>	<p>This option enables the asynchronous mode of operation, where the source code update and the build process will be performed in parallel, instead of waiting for all of the source code updates before starting the build process. This option defaults to enabling asynchronous mode. To disable, set this option to <b>false</b></p> <p>This option is available since the 1.6 release.</p>
<p>binpath</p>	<p>Module setting overrides global</p>	<p>Set this option to set the environment variable PATH while building. You cannot override this setting in a module option. The default value is the \$PATH that is set when the script starts. This environment variable should include the colon-separated paths of your development toolchain. The paths \$KDEDIR /bin and \$QTDIR /bin are automatically added. You may use the tilde (~) for any paths you add using this option.</p>



branch	Module setting overrides global	<p>Set this option to checkout from a branch of KDE instead of the default of <i>master</i> (for Git modules) or <i>trunk</i> (for Subversion), where KDE development occurs.</p> <p>For instance, to checkout KDE 4.6 branch, you would set this option to <code>4.6</code>.</p> <p>If <code>kdesrc-build</code> fails to properly download a branch with this option, you may have to manually specify the URL to download from using the <a href="#">module-base-path</a> or <a href="#">override-url</a> options.</p> <div data-bbox="997 884 1341 1096" style="border: 1px solid black; padding: 5px;"><p><b>NOTE</b></p><p>For most KDE modules you probably wish to use the <a href="#">branch-group</a> option instead and use this option for case-by-case exceptions.</p></div>
--------	---------------------------------	---

branch-group	Module setting overrides global	<p>Set this option to a general group from which you want modules to be chosen. For supported Git module types, kdesrc-build will determine the actual branch to use automatically based on rules encoded by the KDE developers (these rules may be viewed in the <code>kde-build-metadata</code> source repository in your source directory). After a branch is determined that branch is used as if you had specified it yourself using the <code>branch</code> option.</p> <p>This is useful if you're just trying to maintain up-to-date on some normal development track without having to pay attention to all the branch name changes.</p> <p>The current branch groups (as of 2013-08-11) are:</p> <ul style="list-style-type: none"> <li>• <code>stable-qt4</code>, for tracking bugfixes to the Qt™ 4-based KDE libraries and applications.</li> <li>• <code>latest-qt4</code>, for tracking development and new features for the Qt™ 4-based KDE libraries and applications.</li> <li>• <code>kf5-qt5</code>, for tracking 'bleeding-edge' development for the upcoming Qt™ 5-based KDE Frameworks 5, Plasma Workspace 2, etc.</li> </ul> <p>Note that if you <i>do</i> choose a branch yourself, that it will override this setting. The same is true of other specific branch selection options such as <code>tag</code>.</p> <p>This option was added in kdesrc-build 1.16-pre2.</p>
--------------	---------------------------------	--

**NOTE**

This option only applies to `kde-projects` Git modules (the common case). See also `Sec-`

build-dir	Module setting overrides global	<p>Use this option to change the directory to contain the built sources. There are three different ways to use it:</p> <ol style="list-style-type: none"> <li>1. Relative to the KDE Subversion source directory (see <a href="#">the source-dir option</a>). This is the default, and is selected if you type a directory name that does not start with a tilde (~) or a slash (/). The default value is build.</li> <li>2. Absolute path. If you specify a path that begins with a /, then that path is used directly. For example, /tmp/kde-obj-dir/ .</li> <li>3. Relative to your home directory. If you specify a path that begins with a ~, then the path is used relative to your home directory, analogous to the shell's tilde-expansion. For example, ~/builddir would set the build directory to /home/user-name/builddir.</li> </ol> <p>Perhaps surprisingly, this option can be changed per module.</p>
-----------	---------------------------------	---

<p>build-when-unchanged</p>	<p>Module setting overrides global</p>	<p>Use this option in order to control whether kdesrc-build always tries to build a module that has not had any source code updates.</p> <p>By setting <code>build-when-unchanged</code> to <b>true</b>, kdesrc-build always attempts the build phase for a module, even if the module did not have any source code updates. This is the default setting since it is more likely to lead to a correct build.</p> <p>By setting <code>build-when-unchanged</code> to <b>false</b>, kdesrc-build will only attempt to run the build phase for a module if the module has a source code update, or in other situations where it is likely that a rebuild is actually required. This can save time, especially if you run kdesrc-build daily, or more frequently.</p> <div data-bbox="997 1255 1334 1631" style="border: 1px solid black; padding: 5px;"><p><b>IMPORTANT</b></p><p>This feature is provided as an optimization only. Like many other optimizations, there are trade-offs for the correctness of your installation. For instance, changes to the qt or kdelibs modules may cause a rebuild of other modules to be necessary, even if the source code doesn't change at all.</p></div>
-----------------------------	--	--

checkout-only	Module setting overrides global	<p>Set this option to checkout Subversion sources piece by piece. The value for this option should be a space-separated list of directories to checkout. Although this option overrides the global option, be aware that setting this as a global option makes no sense.</p> <p>Note that this setting has no effect on Git modules due to the operation of the Git source control system. See Section <a href="#">6.3.1.1</a> for an example.</p>
cmake-generator	Module setting overrides global	<p>Use this option to specify which generator to use with CMake. Currently both Ninja and Unix Makefiles as well as extra generators based on them like Eclipse CDT4 - Ninja are supported. Invalid (unsupported) values are ignored and treated as if unset. If not set Unix Makefiles will be used by default.</p> <p>Note that if a valid generator is also specified through <a href="#">cmake-options</a> it will override the value for <code>cmake-generator</code>.</p>

cmake-toolchain	Module setting overrides global	<p>Use this option to specify a toolchain file to use with CMake.</p> <p>When a valid toolchain file is configured, kdesrc-build will <i>no longer set environment variables automatically</i>. You can use <a href="#">set-env</a>, <a href="#">binpath</a> and <a href="#">libpath</a> to fix up the environment manually if your toolchain file does not work out of the box with kdesrc-build. Refer to <a href="#">the overview of standard flags added by kdesrc-build</a> for more information.</p> <p>Note that if a valid toolchain is also specified through <a href="#">cmake-options</a> it will override the value for <code>cmake-toolchain</code>.</p>
-----------------	---------------------------------	---

cmake-options	Appends to global options for the default buildsystem, overrides global for other buildsystems.	<p>Use this option to specify what flags to pass to CMake when creating the build system for the module. When this is used as a global option, it is applied to all modules that this script builds. When used as a module option, it is added to the end of the global options. This allows you to specify common CMake options in the global section.</p> <p>This option does not apply to qt (which does not use CMake). Use <a href="#">configure-flags</a> instead.</p> <p>If a valid generator is specified among the listed options it will override the value of <a href="#">cmake-generator</a>. Invalid (unsupported) generators are ignored and will not be passed to CMake.</p> <p>If a valid toolchain file is specified among the listed options it will override the value of <a href="#">cmake-toolchain</a>. Invalid toolchains are ignored and will not be passed to CMake.</p> <p>Since these options are passed directly to the CMake command line, they should be given as they would be typed into CMake. For example:</p> <pre>cmake-options -DCMAKE_BUILD_TYPE=RelWithDebIn</pre> <p>Since this is a hassle, kdesrc-build takes pains to ensure that as long as the rest of the options are set correctly, you should be able to leave this option blank. (In other words, <i>required</i> CMake parameters are set for you automatically)</p>
---------------	---	--

## kdesrc-build Script Manual

colorful-output	Cannot be overridden	Set this option to <b>false</b> to disable the colorful output of kdesrc-build. This option defaults to <i>true</i> . Note that kdesrc-build will not output the color codes to anything but a terminal (such as xterm, Konsole, or the normal Linux <sup>®</sup> console).
compile-commands-export	Module setting overrides global	Enables the generation of a <code>compile_commands.json</code> via CMake inside the build directory. This option defaults to <i>true</i> , set it to <b>false</b> to disable this behavior.
compile-commands-linking	Module setting overrides global	Enables the creation of symbolic links from <code>compile_commands.json</code> generated via CMake inside the build directory to the matching source directory. This option defaults to <i>false</i> , set it to <b>true</b> to enable the automatic symbolic link creation.
configure-flags	Appends to global options for the default buildsystem, overrides global for other buildsystems.	Use this option to specify what flags to pass to <code>./configure</code> when creating the build system for the module. When this is used as a global-option, it is applied to all modules that this script builds. <i>This option only works for qt.</i> To change configuration settings for KDE modules, see <a href="#">cmake-options</a> .



<p>custom-build-command</p>	<p>Module setting overrides global (build system option)</p>	<p>This option can be set to run a different command (other than <b>make</b>, for example) in order to perform the build process. kdesrc-build should in general do the right thing, so you should not need to set this option. However it can be useful to use alternate build systems. The value of this option is used as the command line to run, modified by the <a href="#">make-options</a> option as normal.</p>
<p>cxxflags</p>	<p>Appends to global options for the default buildsystem, overrides global for other buildsystems.</p>	<p>Use this option to specify what flags to use for building the module. This option is specified here instead of with <a href="#">configure-flags</a> or <a href="#">cmake-options</a> because this option will also set the environment variable <code>CXXFLAGS</code> during the build process. Note that for KDE 4 and any other modules that use CMake, it is necessary to set the <code>CMAKE_BUILD_TYPE</code> option to <b>none</b> when configuring the module. This can be done using the <a href="#">cmake-options</a> option.</p>
<p>dest-dir</p>	<p>Module setting overrides global</p>	<p>Use this option to change the name a module is given on disk. For example, if your module was <code>extragear/network</code>, you could rename it to <code>extragear-network</code> using this option. Note that although this changes the name of the module on disk, it is not a good idea to include directories or directory separators in the name as this will interfere with any <a href="#">build-dir</a> or <a href="#">source-dir</a> options.</p>

## kdesrc-build Script Manual

disable-agent-check	Cannot be overridden	Normally if you are using SSH to download the Subversion sources (such as if you are using the svn+ssh protocol), kdesrc-build will try and make sure that if you are using ssh-agent, it is actually managing some SSH identities. This is to try and prevent SSH from asking for your pass phrase for every module. You can disable this check by setting <code>disable-agent-check</code> to <b>true</b> .
do-not-compile	Module setting overrides global	Use this option to select a specific set of directories not to be built in a module (instead of all of them). The directories not to build should be space-separated. Note that the sources to the programs will still be downloaded. You can use the <code>checkout-only</code> directive to choose directories that you want to check out. For example, to hold JuK and KsCD in the <code>kdemultimedia</code> module from compiling, you would add <code>"do-not-compile juk kscd"</code> to your <code>kdemultimedia</code> settings. See Section 6.3.1.2 for an example.
email-address	Cannot be overridden	This option was removed in kdesrc-build 1.14.
email-on-compile-error	Cannot be overridden	This option was removed in kdesrc-build 1.14.
inst-apps		This option was removed in version 1.10

git-desired-protocol	Cannot be overridden	<p>This option only applies to modules from a <a href="#">KDE project</a> repository. What this option actually does is configure which network protocol to prefer when pushing source code for these modules. Normally the very-efficient <code>git</code> protocol is used, but this may be blocked in some networks (e.g. corporate intranets, public Wi-Fi). An alternative protocol which is much better supported is the <code>https</code> protocol used for Internet web sites. If you are using one of these constrained networks you can set this option to <b>http</b> to prefer <code>https</code> communications instead.</p> <div data-bbox="997 1031 1341 1205" style="border: 1px solid black; padding: 5px;"> <p><b>TIP</b> You may also need the <a href="#">http-proxy</a> option if an HTTP proxy is also needed for network traffic.</p> </div> <p>In any other situation you should not set this option as the default protocol is most efficient. This option was added in <code>kdesrc-build</code> 1.16. Prior to 20.06 this option was used to configure the fetch URL instead of the push URL. As of 20.06 <code>https</code> is always used when updating KDE projects.</p>
----------------------	----------------------	---

git-repository-base

Cannot be overridden

This option, added in version 1.12.1, is used to create a short name to reference a specific Git repository base URL in later [module set](#) declarations, which is useful for quickly declaring many Git modules to build. You must specify two things (separated by a space): The name to assign to the base URL, and the actual base URL itself. For example:

```
global
  # other options

  # This is the common path to all anonymous C
  git-repository-base kde-git kde:
end global

# Module declarations

module-set
  # Now you can use the alias you defined earl
  # in a module-set.
  repository kde-git

  use-modules module1.git module2.git
end module-set
```

The `module-set's` `use-modules` option created two modules internally, with `kdesrc-build` behaving as if it had read:

```
module module1
  repository kde:module1.git
end module

module module2
  repository kde:module2.git
end module
```

The `kde:` Git repository prefix used above is a shortcut which will be setup by `kdesrc-build` automatically. See the [TechBase URL Renaming](#) article for more information. Note that unlike most other options, this option can be specified multiple times in order to create as many

<p>git-user</p>	<p>Module setting overrides global</p>	<p>This option is intended for KDE developers. If set, it will be used to automatically setup identity information for the Git source control software for <i>newly downloaded</i> Git modules (including the vast majority of KDE modules). Specifically, the user's name and email fields for each new Git repository are filled in to the values set by this option. The value must be specified in the form <code>User Name &lt;email@example.com&gt;</code>. For instance, a developer named 'Foo Barbaz' with the email address 'foo@abc.xyz' would use:</p> <pre>git-user Foo Barbaz &lt;foo@abc.xyz&gt;</pre> <p>This option was introduced in kdesrc-build 15.09.</p>
<p>http-proxy</p>	<p>Module setting overrides global</p>	<p>This option, if set, uses the specified URL as a proxy server to use for any HTTP network communications (for example, when downloading snapshots for new modules, or the <a href="#">KDE project database</a>). In addition, kdesrc-build will try to ensure that the tools it depends on also use that proxy server, if possible, by setting the <code>http_proxy</code> environment variable to the indicated server, <i>if that environment variable is not already set</i>. This option was introduced with kdesrc-build 1.16.</p>
<p>ignore-kde-structure</p>	<p>Module setting overrides global</p>	<p>This option is deprecated and will be removed (again) in a future release. Please use <code>directory-layout</code> instead, which offers more options.</p>

directory-layout	Module setting overrides global	<p>This option is used to configure the layout which kdesrc-build should use when creating source and build directories.</p> <p>Currently, there are three possible values: <b>metadata</b>, <b>invent</b> and <b>flat</b></p> <p>The <b>flat</b> layout will group all modules directly underneath the top level source and build directories. For example, source/extragear/network/telepathy/ktp-text-ui in the <b>metadata</b> layout would be source/ktp-text-ui using the <b>flat</b> layout instead.</p> <p>The <b>invent</b> layout creates a directory hierarchy mirroring the relative paths of repositories on <a href="http://invent.kde.org">invent.kde.org</a>. For example source/kde/applications/kate in the <b>metadata</b> layout would be source/utilities/kate using the <b>invent</b> layout instead. This layout only affects KDE projects. It is a good choice for people starting out with kdesrc-build.</p> <p>Finally, the <b>metadata</b> layout is the same as the old default behaviour. This layout organises KDE projects according to the project paths specified in the project metadata for these modules. This is a good choice if you want a directory layout which tracks with certain KDE processes, but note that this path is therefore not always stable. As a result, kdesrc-build may abandon an old copy of the repository and clone a new one for a project due to changes in the project metadata.</p> <p>By default the <b>metadata</b> layout will be used, to preserve backwards</p>
------------------	---------------------------------	--

ignore-modules	Can't be overridden	<p>Modules named by this option, which would be chosen by kdesrc-build due to a <a href="#">use-modules</a> option, are instead skipped entirely. Use this option when you want to build an entire <a href="#">kde-projects</a> project grouping <i>except for</i> some specific modules. The option value does not necessarily have to name the module directly. Any module that has full consecutive parts of its <a href="#">KDE projects module path</a> match one of the option values will be ignored, so you can ignore multiple modules this way. For example, an option value of <i>libs</i> would result in both <code>kde/kdegraphics/libs</code> and <code>playground/libs</code> being excluded (though not <code>kde/kdelibs</code> since the full part 'kdelibs' is what is compared).</p> <div data-bbox="992 1276 1339 1360" style="border: 1px solid black; padding: 5px;"> <p><b>TIP</b> See also <a href="#">Example 2.7</a>.</p> </div> <p>This option was introduced with kdesrc-build 1.16.</p>
----------------	---------------------	--

<p>include-dependencies</p>	<p>Module setting overrides global</p>	<p>This option, when set to <b>true</b> requests that kdesrc-build also include known dependencies of this module in its build, without requiring you to mention those dependencies (even indirectly).</p> <div data-bbox="997 590 1341 831" style="border: 1px solid black; padding: 5px;"> <p><b>NOTE</b> This option only works for <i>kde-project-based modules</i>, and requires that the metadata maintained by the KDE developers is accurate for your selected <i>branch-group</i>.</p> </div> <p>This option is enabled by default, to support building applications that need versions of Qt™ or Plasma more recent than supported on common operating systems.</p>
<p>install-after-build</p>	<p>Module setting overrides global</p>	<p>This option is used to install the package after it successfully builds. This option is enabled by default. If you want to disable this, you need to set this option to <b>false</b> in the <i>configuration file</i>. You can also use the <code>--no-install</code> command line flag.</p>



<p>install-environment-driver</p>	<p>Cannot be overridden</p>	<p>By default, kdesrc-build will install a shell script that can be sourced in a user's profile setup scripts to easily establish needed environment variables to run the Plasma desktop built by kdesrc-build. This driver will alter the following files:</p> <ul style="list-style-type: none"> <li>• <code>\$XDG_CONFIG_HOME/kde-env-master.sh</code> (normally found at <code>~/.config/kde-env-master.sh</code>).</li> <li>• <code>\$XDG_CONFIG_HOME/kde-env-user.sh</code> (normally found at <code>~/.config/kde-env-user.sh</code>).</li> </ul> <p>The <code>kde-env-user.sh</code> is optional. It is intended for user customizations (see the <a href="#">Troubleshooting and Debugging</a> section of the KDE UserBase for examples of customizable settings), but these settings can be set elsewhere by the user in their existing profile setup scripts. You can disable this feature by setting this option to <code>false</code>, and ensuring that the <a href="#">install-session-driver</a> option is also disabled. This option was introduced with kdesrc-build 17.08.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p><b>TIP</b> kdesrc-build will not overwrite your existing files (if present) unless you also pass the <code>--delete-my-settings</code> command-line option.</p> </div>
-----------------------------------	-----------------------------	--

<p>install-session-driver</p>	<p>Cannot be overridden</p>	<p>If enabled, kdesrc-build will try to install a driver for the graphical login manager that allows you to login to your kdesrc-build-built KDE desktop. This driver will alter the following files:</p> <ul style="list-style-type: none"> <li>• <code>~/.xsession</code></li> <li>• <code>\$XDG_CONFIG_HOME/kde-env-master.sh</code> (normally found at <code>~/.config/kde-env-master.sh</code>).</li> <li>• <code>\$XDG_CONFIG_HOME/kde-env-user.sh</code> (normally found at <code>~/.config/kde-env-user.sh</code>).</li> </ul> <p>If you maintain your own login driver then you can disable this feature by setting this option to <code>false</code>. If enabled, this feature also enables the <a href="#">install-environment-driver</a> feature.</p> <p>This option was introduced with kdesrc-build 1.16.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p><b>TIP</b> kdesrc-build will not overwrite your existing files (if present) unless you also pass the <code>--delete-my-settings</code> command-line option.</p> </div>
<p>kdedir</p>	<p>Module setting overrides global</p>	<p>This option sets the directory that KDE will be installed to after it is built. It defaults to <code>~/kde</code>. If you change this to a directory needing root access, you may want to read about the <a href="#">make-install-prefix</a> option as well.</p>

kde-languages	Cannot be overridden	<p>This option allows you to choose to download and install localization packages along with KDE. You might do this if you do not live in the United States and would like to use KDE translated into your native language. To use this option, set it to a space-separated list of languages to install. Each language has a language code associated with it, which you can look up at this page: <a href="http://110n.kde.org/teams-list.php">http://110n.kde.org/teams-list.php</a>. It is alright to choose only one language. By default, none are downloaded, which means KDE will display in American English. For instance, to choose to install French, you would set the option to something like: <b>kde-languages fr</b>. You would still need to use System Settings in order to choose the French language, however.</p>
libpath	Module setting overrides global	<p>Set this option to set the environment variable <code>LD_LIBRARY_PATH</code> while building. You cannot override this setting in a module option. The default value is blank, but the paths <code>\$KDEDIR /lib</code> and <code>\$QTDIR /lib</code> are automatically added. You may use the tilde (<code>~</code>) for any paths you add using this option.</p>
log-dir	Module setting overrides global	<p>Use this option to change the directory used to hold the log files generated by the script.</p>

## kdesrc-build Script Manual

make-install-prefix	Module setting overrides global	Set this variable to a space-separated list, which is interpreted as a command and its options to precede the <b>make install</b> command used to install modules. This is useful for installing packages with Sudo for example, but please be careful while dealing with root privileges.
make-options	Module setting overrides global (build system option)	Set this variable in order to pass command line options to the <b>make</b> command. This is useful for programs such as <a href="#">distcc</a> or systems with more than one processor core. Note that not all supported build systems use <b>make</b> . For build systems that use <b>ninja</b> for build (such as the <a href="#">Meson build system</a> ), see the <a href="#">ninja-options</a> setting.
manual-build	Module setting overrides global	Set the option value to <b>true</b> to keep the build process from attempting to build this module. It will still be kept up-to-date when updating from Subversion. This option is exactly equivalent to the <code>--no-build</code> command line option.
manual-update	Module setting overrides global	Set the option value to <b>true</b> to keep the build process from attempting to update (and by extension, build or install) this module. If you set this option for a module, then you have essentially commented it out.

## kdesrc-build Script Manual

<p>module-base-path</p>	<p>Module setting overrides global</p>	<p>Set this option to override kdesrc-build's default directory path to the module in question. This can be used, for example, to pull specific branches or tagged versions of libraries. <a href="#">The KDE Source Viewer</a> is invaluable in helping to pick the right path. Note that kdesrc-build constructs the final path according to the following template: <code>\$svn-server /home/kde/ \$module-base-path</code>. The default value is either <code>trunk/ \$module</code> or <code>trunk/KDE/ \$module</code>, depending on the module name.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p><b>TIP</b> Use the <a href="#">branch</a> or <a href="#">tag</a> options instead whenever they are applicable.</p> </div>
<p>niceness</p>	<p>Cannot be overridden</p>	<p>Set this option to a number between 20 and 0. The higher the number, the lower a priority kdesrc-build will set for itself, i.e. the higher the number, the "nicer" the program is. The default is 10.</p>

<p>ninja-options</p>	<p>Module setting overrides global (build system option)</p>	<p>Set this variable in order to pass command line options to the <b>ninja</b> build command. This can be useful to enable 'verbose' output or to manually reduce the number of parallel build jobs that <b>ninja</b> would use.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p><b>NOTE</b></p> <p>Note that this setting only controls ninja when used by kdesrc-build. The Qt™ 'webengine' module uses <b>ninja</b> indirectly, but only officially supports being built by <b>make</b>. In this situation, you can set <code>NINJAFLAGS</code> as a way to have <b>make</b> pass the appropriate flags when it later calls <b>ninja</b>, by using <a href="#">make-options</a>.</p> <pre style="font-family: monospace; font-size: 0.9em;">options qtwebengine     # Restrict make and ninja to using no more     # when more CPU is available, to avoid run     make-options -j6 NINJAFLAGS=-j6 end options</pre> </div>
<p>no-svn</p>	<p>Module setting overrides global</p>	<p>If this option is set to true then kdesrc-build will not update the source code for the module automatically. It will still try to build the module if it normally would have tried anyways.</p>
<p>no-rebuild-on-fail</p>		<p>This option was removed in version 1.10, since this behavior no longer helps due to fixes in the underlying build system.</p>

## kdesrc-build Script Manual

<p>num-cores</p>	<p>Cannot be overridden</p>	<p>This option is defined by kdesrc-build (when using the kdesrc-build-setup tool or <b>kdesrc-build --initial-setup</b>), set to be the number of available CPUs (as indicated by the external application nproc). If kdesrc-build cannot detect the number of CPUs, this value is set to 4. See Example 2.1 for an example of this option's usage. This option was added in version 20.07.</p>
<p>num-cores-low-mem</p>	<p>Cannot be overridden</p>	<p>This option is defined by kdesrc-build (when using the kdesrc-build-setup tool or <b>kdesrc-build --initial-setup</b>), set to be the number of CPUs that is deemed safe for heavyweight or other highly-intensive modules, such as qtwebengine, to avoid running out of memory during the build. The typical calculation is one CPU core for every 2 gigabytes (GiB) of total memory. At least 1 core will be specified, and no more than num-cores cores will be specified. Although this option is intended to support Qt™ modules, you can use it for your any module in the same way that num-cores is used. If kdesrc-build cannot detect available memory then this value will be set to 2. This option was added in version 20.07.</p>

	<p>Module setting overrides global</p>	<p>This is an advanced option, added in kdesrc-build 1.16. Normally kdesrc-build will detect the appropriate build system to use for a module after it is downloaded. This is done by checking for the existence of specific files in the module's source directory. Some modules may include more than one required set of files, which could confuse the auto-detection. In this case you can manually specify the correct build type. Currently supported build types that can be set are:</p> <p><b>KDE</b></p> <p>Used to build KDE modules. In reality it can be used to build almost any module that uses CMake but it is best not to rely on this.</p> <p><b>Qt</b></p> <p>Used to build the Qt™ library itself.</p> <p><b>qmake</b></p> <p>Used to build Qt™ modules that use qmake-style .pro files.</p> <p><b>generic</b></p> <p>Used to build modules that use plain Makefiles and that do not require any special configuration.</p> <p><b>autotools</b></p> <p>This is the standard configuration tool used for most Free and open-source software not in any of the other categories.</p>
override-build-system		



## kdesrc-build Script Manual

override-url	Module setting overrides global	If you set this option, kdesrc-build will use its value as the URL to pass to Subversion <i>completely unchanged</i> . You should generally use this if you want to download a specific release but kdesrc-build cannot figure out what you mean using <a href="#">branch</a> .
persistent-data-file	Cannot be overridden	Use this option to change where kdesrc-build stores its persistent data. The default is to store this data in a file called <code>.kdesrc-build-data</code> placed in the same directory as the configuration file in use. If you have multiple available configurations in the same directory you may want to manually set this option so that the different configurations do not end up with conflicting persistent data. This option was added with kdesrc-build 1.15.
prefix	Module setting overrides global	This option controls where to install the module (normally the <code>kdedir</code> setting is used). Using this option allows you to install a module to a different directory than where the KDE Platform libraries are installed, such as if you were using kdesrc-build only to build applications. You can use <code>\${MODULE}</code> or <code>\$MODULE</code> in the path to have them expanded to the module's name.
purge-old-logs	Module setting overrides global	This option controls whether old log directories are automatically deleted or not. The default value is <code>true</code> .

qmake-options	Module setting overrides global	Any options specified here are passed to the <b>qmake</b> command, for modules that use the qmake build system. For instance, you can use the <b>PREFIX=/path/to/qt</b> option to qmake to override where it installs the module. This option was added to kdesrc-build 1.16.
qtdir	Module setting overrides global	Set this option to set the environment variable <code>QTDIR</code> while building. If you do not specify this option, kdesrc-build will assume that Qt™ is provided by the operating system.
remove-after-install	Module setting overrides global	<p>If you are low on hard disk space, you may want to use this option in order to automatically delete the build directory (or both the source and build directories for one-time installs) after the module is successfully installed.</p> <p>Possible values for this option are:</p> <ul style="list-style-type: none"> <li>• none - Do not delete anything (This is the default).</li> <li>• builddir - Delete the build directory, but not the source.</li> <li>• all - Delete both the source code and build directory.</li> </ul> <p>Note that using this option can have a significant detrimental impact on both your bandwidth usage (if you use <i>all</i>) and the time taken to compile KDE software, since kdesrc-build will be unable to perform incremental builds.</p>

<p>repository</p>	<p>Module setting overrides global</p>	<p>This option was introduced with version 1.10, and is used to specify the Git repository to download the source code for the module. Qt™ (and therefore qt) would need this option, as well as various KDE modules that are in the process of conversion to use Git.</p>
<p>revision</p>	<p>Module setting overrides global</p>	<p>If this option is set to a value other than 0 (zero), kdesrc-build will force the source update to bring the module to the exact revision given, even if options like <a href="#">branch</a> are in effect. If the module is already at the given revision then it will not be updated further unless this option is changed or removed from the configuration.</p> <div data-bbox="997 1094 1338 1272" style="border: 1px solid black; padding: 5px;"> <p><b>NOTE</b> This option did not work for git-based modules (including <a href="#">kde-projects</a> modules) until kdesrc-build version 1.16.</p> </div>
<p>run-tests</p>	<p>Module setting overrides global (build system option)</p>	<p>If set to <b>true</b>, then the module will be built with support for running its test suite, and the test suite will be executed as part of the build process. kdesrc-build will show a simple report of the test results. This is useful for developers or those who want to ensure their system is setup correctly.</p>

set-env	Module setting overrides global	<p>This option accepts a space-separated set of values, where the first value is the environment variable to set, and the rest of the values is what you want the variable set to. For example, to set the variable <code>RONALD</code> to McDonald, you would put in the appropriate section this command:</p> <pre>set-env RONALD McDonald</pre> <p>This option is special in that it can be repeated without overriding earlier set-env settings in the same section of the <a href="#">configuration file</a>. This way you can set more than one environment variable per module (or globally).</p>
source-dir	Module setting overrides global	<p>This option is used to set the directory on your computer to store the KDESubversion sources at. If you do not specify this value, the default is <code>~/kdesrc</code>. You may use the tilde (<code>~</code>) to represent the home directory if using this option.</p>
ssh-identity-file	Cannot be overridden	<p>Set this option to control which private SSH key file is passed to the <code>ssh-add</code> command when <code>kdesrc-build</code> is downloading source code from repositories that require authentication. See also: Section <a href="#">6.4.1</a>. This option was added in version 1.14.2.</p>
stop-on-failure	Module setting overrides global	<p>Set this option value to <code>true</code> to cause the script to stop execution after an error occurs during the build or install process. This option is off by default.</p>

<p>svn-server</p>	<p>Module setting overrides global</p>	<p>This option is used to set the server used to check out from Subversion. The default is the anonymous Subversion repository, <code>svn://anonsvn.kde.org/</code></p> <div data-bbox="997 527 1336 768" style="border: 1px solid black; padding: 5px;"> <p><b>NOTE</b> If you are developing for KDE, use the Subversion repository that was provided to you when you received your developer account, instead of the anonymous repository.</p> </div>
<p>tag</p>	<p>Module setting overrides global</p>	<p>Use this option to download a specific release of a module. <i>Note:</i> The odds are very good that you <i>do not want</i> to use this option. KDE releases are available in tarball form from the <a href="#">KDE download site</a>.</p> <div data-bbox="997 1115 1336 1255" style="border: 1px solid black; padding: 5px;"> <p><b>NOTE</b> This option has only been supported for git-based modules since kdesrc-build 1.16.</p> </div>
<p>use-clean-install</p>	<p>Module setting overrides global (build system option)</p>	<p>Set this option to <b>true</b> in order to have kdesrc-build run <b>make uninstall</b> directly before running <b>make install</b>. This can be useful in ensuring that there are not stray old library files, CMake metadata, etc. that can cause issues in long-lived KDE installations. However this only works on build systems that support <b>make uninstall</b>. This option was added with kdesrc-build 1.12, but was not documented until kdesrc-build 1.16.</p>

## kdesrc-build Script Manual

use-cmake		This option was removed in kdesrc-build 1.4 as all KDE 4 modules require CMake, and CMake use is not permitted on any other modules.
use-idle-io-priority	Cannot be overridden	This option, added in kdesrc-build 1.12, will cause a lower priority to be used for disk and other I/O usage, which can significantly improve the responsiveness of the rest of the system at the expense of slightly longer running times for kdesrc-build. The default is to be disabled, to enable the lower disk priority set this to <b>true</b> .
use-inactive-modules	Cannot be overridden	This option when enabled will allow kdesrc-build to also clone and pull from repositories marked as inactive. The default is to be disabled, to allow inactive modules set this to <b>true</b> .

use-modules

Can only use in [module-set](#)

This option, added in kdesrc-build 1.12.1, allows you to easily specify many different modules to build at the same point in [the configuration file](#).

This option *must* be used within a `module-set`. Every identifier passed to this option is internally converted to a kdesrc-build module, with a [repository](#) option set to the `module-set`'s repository combined with the identifier name in order to setup the final repository to download from. All other options that are assigned in the `module-set` are also copied to the generated modules unaltered.

The order that modules are defined in this option is important, because that is also the order that kdesrc-build will process the generated modules when updating, building, and installing. All modules defined in the given `module-set` will be handled before kdesrc-build moves to the next module after the `module-set`.

If you need to change the options for a generated module, simply declare the module again after it is defined in the `module-set`, and set your options as needed. Although you will change the options set for the module this way, the module will still be updated and built in the order set by the `module-set` (i.e. you can't reorder the build sequence doing this).

**IMPORTANT**

The name to use for the module if you do this is simply the name that you passed to `use-modules`, with the exception that any `.git` is removed.

## kdesrc-build Script Manual

use-qt-builddir-hack	Module setting overrides global	This option has been removed due to improvements in the Qt™ build system.
use-stable-kde	Can't be overridden	This option is deprecated and will be removed (again) in a future release. Please use the <code>branch-group</code> instead, which offers more selectivity.

Table 4.1: Table of Options



## Chapter 5

# Command Line Options and Environment Variables

### 5.1 Command Line Usage

kdesrc-build is designed to be run as follows:

```
kdesrc-build [--options...] [modules to build...]
```

If no modules to build are specified on the command line, then kdesrc-build will build all modules defined in its configuration file, in the order listed in that file (although this can be modified by various configuration file options).

#### 5.1.1 Commonly used command line options

The full list of command line options is given in Section 5.3. The most-commonly used options include:

##### **--pretend (or -p)**

This option causes kdesrc-build to indicate what actions it would take, without actually really implementing them. This can be useful to make sure that the modules you think you are building will actually get built.

##### **--refresh-build**

This option forces kdesrc-build to build the given modules from an absolutely fresh start point. Any existing build directory for that module is removed and it is rebuilt. This option is useful if you have errors building a module, and sometimes is required when Qt™ or KDE libraries change.

##### **--no-src**

This option skips the source update process. You might use it if you have very recently updated the source code (perhaps you did it manually or recently ran kdesrc-build) but still want to rebuild some modules.

##### **--no-build**

This option is similar to `--no-src` above, but this time the build process is skipped.

### 5.1.2 Specifying modules to build

In general, specifying modules to build is as simple as passing their module name as you defined it in the configuration file. You can also pass modules that are part of a module set, either as named on [use-modules](#), or the name of the entire module set itself, if you have given it a name.

In the specific case of module sets based against the [KDE project database](#), kdesrc-build will expand module name components to determine the exact module you want. For example, kdesrc-build's KDE project entry locates the project in `extragear/utils/kdesrc-build`. You could specify any of the following to build kdesrc-build:

```
% kdesrc-build +extragear/utils/kdesrc-build
% kdesrc-build +utils/kdesrc-build
% kdesrc-build +kdesrc-build
```

#### NOTE

The commands in the previous example preceded the module-name with a `+`. This forces the module name to be interpreted as a module from the KDE project database, even if that module hasn't been defined in your configuration file.

Be careful about specifying very generic projects (e.g. `extragear/utils` by itself), as this can lead to a large amount of modules being built. You should use the `--pretend` option before building a new module set to ensure it is only building the modules you want.

## 5.2 Supported Environment Variables

kdesrc-build does not use environment variables. If you need to set environment variables for the build or install process, please see the [set-env](#) option.

## 5.3 Supported command-line parameters

The script accepts the following command-line options:

#### **`--async`**

Enables the [asynchronous mode](#), which can perform the source code updates and module builds at the same time. This is the default, this option only needs specified if you have disabled it in the configuration.

#### **`--help`**

Only display simple help on this script.

#### **`--version` (or `-v`)**

Display the program version.

#### **`--show-info`**

Displays information about kdesrc-build and the operating system, that may prove useful in bug reports or when asking for help in forums or mailing lists.

Available since version 18.11.

## kdesrc-build Script Manual

### **--initial-setup**

Has kdesrc-build perform the one-time initial setup necessary to prepare the system for kdesrc-build to operate, and for the newly-installed KDE software to run.

This includes:

- Installing known dependencies (on supported Linux<sup>®</sup> distributions)
- Adding required environment variables to `~/.bashrc`
- Setting up a [configuration file](#)

Available since version 18.11.

### **--author**

Display contact information for the author.

### **--color**

Enable colorful output. (This is the default for interactive terminals).

### **--nice=value**

This value adjusts the computer CPU priority requested by kdesrc-build, and should be in the range of 0-20. 0 is highest priority (because it is the least 'nice'), 20 is lowest priority. kdesrc-build defaults to 10.

### **--no-async**

Disables the [asynchronous mode](#) of updating. Instead the update will be performed in its entirety before the build starts. This option will slow down the overall process, but if you encounter IPC errors while running kdesrc-build try using this option, and submitting a [bug report](#).

### **--no-color**

Disable colorful output.

### **--pretend (or -p)**

kdesrc-build will run through the update and build process, but instead of performing any actions to update or build, will instead output what the script would have done (e.g. what commands to run, general steps being taken, etc.).

#### NOTE

Simple read-only commands (such as reading file information) may still be run to make the output more relevant (such as correctly simulating whether source code would be checked out or updated).

#### IMPORTANT

This option requires that some needed metadata is available, which is normally automatically downloaded, but downloads are disabled in pretend mode. If you've never run kdesrc-build (and therefore, don't have this metadata), you should run **kdesrc-build --metadata-only** to download the required metadata first.

### **--quiet (or -q)**

Do not be as noisy with the output. With this switch only the basics are output.

### **--really-quiet**

Only output warnings and errors.

### **--verbose**

Be very descriptive about what is going on, and what kdesrc-build is doing.

## kdesrc-build Script Manual

### **`--src-only` (or `--svn-only`)**

Only perform the source update. (The `--svn-only` is only supported for compatibility with older scripts).

### **`--build-only`**

Only perform the build process.

### **`--install-only`**

Only perform the install process.

### **`--metadata-only`**

Only perform the metadata download process. `kdesrc-build` normally handles this automatically, but you might manually use this to allow the `--pretend` command line option to work.

### **`--rebuild-failures`**

Use this option to build only those modules which failed to build on a previous `kdesrc-build` run. This is useful if a significant number of failures occurred mixed with successful builds. After fixing the issue causing the build failures you can then easily build only the modules that failed previously.

#### **NOTE**

Note that the list of 'previously-failed modules' is reset every time a `kdesrc-build` run finishes with some module failures. However it is not reset by a completely successful build, so you can successfully rebuild a module or two and this flag will still work.

This option was added for `kdesrc-build` 15.09.

### **`--include-dependencies`, `--no-include-dependencies`**

This option causes `kdesrc-build` to automatically include other KDE and Qt™ modules in the build, if required for the modules you have requested to build on the command line or in your [configuration file](#).

The modules that are added are as recorded within the KDE source code management system. See Section [2.6.4](#).

The corresponding configure file option is [include-dependencies](#).

You can also use `--no-include-dependencies`, which turns off automatic inclusion of additional dependency modules.

### **`--ignore-modules`**

Do not include the modules passed on the rest of the command line in the update/build process (this is useful if you want to build most of the modules in your [configuration file](#) and just skip a few).

### **`--no-src` (or `--no-svn`)**

Skip contacting the Subversion server. (The `--no-svn` parameter is only supported for compatibility with older versions of the script).

### **`--no-build`**

Skip the build process.

### **`--no-metadata`**

Do not automatically download the extra metadata needed for KDE git modules. The source updates for the modules themselves will still occur unless you pass `--no-src` as well.

This can be useful if you are frequently re-running `kdesrc-build` since the metadata does not change very often. But note that many other features require the metadata to be available. You might want to consider running `kdesrc-build` with the `--metadata-only` option one time and then using this option for subsequent runs.

## kdesrc-build Script Manual

### ***--no-install***

Do not automatically install packages after they are built.

### ***--no-build-when-unchanged, --force-build***

This option explicitly disables skipping the build process (an optimization controlled by the [build-when-unchanged](#) option). This is useful for making kdesrc-build run the build when you have changed something that kdesrc-build cannot check.

*--force-build* performs the exact same function, and is perhaps easier to remember.

### ***--debug***

Enables debug mode for the script. Currently this means that all output will be dumped to the standard output in addition to being logged in the log directory like normal. Also, many functions are much more verbose about what they are doing in debugging mode.

### ***--query=mode***

This command causes kdesrc-build to query a parameter of the modules in the build list (either passed on the command line or read in from the configuration file), outputting the result to screen (one module per line).

This option must be provided with a 'query mode', which should be one of the following:

- *source-dir*, which causes kdesrc-build to output the full path to where the module's source code is stored.
- *build-dir*, which causes kdesrc-build to output the full path to where the module build process occurs.
- *install-dir*, which causes kdesrc-build to output the full path to where the module will be installed.
- *project-path*, which causes kdesrc-build to output the location of the module within the hierarchy of KDE source code repositories. See [Section 2.6.4](#) for more information on this hierarchy.
- *branch*, which causes kdesrc-build to output the resolved git branch that will be used for each module, based on the [tag](#), [branch](#) and [branch-group](#) settings in effect.
- Otherwise, option names that are valid for modules in the [configuration file](#) can be used, the resolved value of which will be listed for each module.

If a single module is passed on the command line, then the output is simply the value of the parameter being queried. If multiple (or no) modules are passed on the command line, then each line is prefixed by the name of the module. Either way, kdesrc-build stops running once each value is output.

This option was added with kdesrc-build 16.05.

For example, the command '**kdesrc-build --query branch kactivities kdepim**' might end up with output like:

```
kactivities: master
kdepim: master
```

### ***--no-rebuild-on-fail***

Do not try to rebuild modules that have failed building from scratch. kdesrc-build will never try to do this to a module that already was tried to be built from scratch.

### ***--refresh-build***

Recreate the build system and make from scratch.

### ***--reconfigure***

Run **cmake** (for KDE modules) or **configure** (for Qt™) again, without cleaning the build directory. You should not normally have to specify this, as kdesrc-build will detect when you change the relevant options and automatically re-run the build setup. This option is implied if *--refresh-build* is used.

**--resume-from**

This option is used to resume the build starting from the given module, which should be the next option on the command line. You should not specify other module names on the command line.

**NOTE**

This option formerly added `--no-src`, but does not any longer (since kdesrc-build 1.13). If you want to avoid source updates when resuming, simply pass `--no-src` in addition to the other options.

See also: `--resume-after` and Section 6.3.5.1. You would prefer to use this command line option if you have fixed the build error and want kdesrc-build to complete the build.

**--resume-after**

This option is used to resume the build starting after the given module, which should be the next option on the command line. You should not specify other module names on the command line.

**NOTE**

This option formerly added `--no-src`, but does not any longer (since kdesrc-build 1.13). If you want to avoid source updates when resuming, simply pass `--no-src` in addition to the other options.

See also: `--resume-from` and Section 6.3.5.1. You would prefer to use this command line option if you have fixed the build error and have also built and installed the module yourself, and want kdesrc-build to start again with the next module.

**--resume**

This option can be used to run kdesrc-build after it has had a build failure.

It resumes the build from the module that failed, using the list of modules that were waiting to be built before, and disables source and metadata updates as well. The use case is when a simple mistake or missing dependency causes the build failure. Once you correct the error you can quickly get back into building the modules you were building before, without fiddling with `--resume-from` and `--stop-before`.

This is even handier with the `--stop-on-failure` command line argument, especially if you're initially setting up your development environment.

This option was added with kdesrc-build 1.16.

**--stop-before**

This command line option is used to stop the normal build process just *before* a module would ordinarily be built.

For example, if the normal build list was `moduleA`, `moduleB`, `moduleC`, then `--stop-before=moduleB` would cause kdesrc-build to only build `moduleA`.

This command line option was added with kdesrc-build 1.16.

**--stop-after**

This command line option is used to stop the normal build process just *after* a module would ordinarily be built.

For example, if the normal build list was `moduleA`, `moduleB`, `moduleC`, then `--stop-after=moduleB` would cause kdesrc-build to build `moduleA` and `moduleB`.

This command line option was added with kdesrc-build 1.16.

**--stop-on-failure**

This option causes the build to abort as soon as a failure occurs. Useful when you're setting up your initial development environment. Without this flag, kdesrc-build will try to press on with the rest of the modules in the build to avoid wasting time in case the problem is with a single module.

This option was added with kdesrc-build 1.16. See also the [stop-on-failure](#) option.

**--rc-file**

This interprets the next command line parameter as the file to read the configuration options from. The default value for this parameter is `kdesrc-buildrc` (checked in the current directory) if it is present, or `~/kdesrc-buildrc` otherwise. See also [chapter 4](#).

**--print-modules**

Takes all actions up to and including dependency reordering of the modules specified on the command line (or configuration file), prints the modules that would be processed one per line, and then exits without further action.

The `kde-project` metadata is downloaded first (though, see [--pretend](#) or [--no-src](#)).

The output is not fully compatible with usage by scripts as other output messages may be generated until the module list is shown.

This is mostly just useful for quickly determining what kdesrc-build understands a module's dependencies to be, which means it's only useful for [kde-projects](#) modules. This option is also compatible with [--resume-from](#), [--resume-after](#), [--stop-before](#), [--stop-after](#).

**--list-build**

Lists the modules that would be built, in the order in which they would be built. If applicable, the output listing also mentions which commit/branch/tag would be selected for checkout.

This option is similar to [--print-modules](#). For more detailed information on how modules relate to each other, see also: [--dependency-tree](#).

**--dependency-tree**

Prints out dependency information on the modules that would be built using a tree format (recursive). Listed information also includes which specific commit/branch/tag is depended on and whether or not the dependency would be built. Note: the generated output may become quite large for applications with many dependencies.

**--run**

This option interprets the next item on the command line as a program to run, and kdesrc-build will then finish reading the configuration file, update the environment as normal, and then execute the given program.

This will not work to start a shell with the kdesrc-build environment in most cases however, since interactive shells typically reset at least part of the environment variables (such as `PATH` and `KDEDIRS`) in the startup sequence.

**TIP**

If you want to see the environment used by kdesrc-build, you can run the **printenv** command:

```
$ kdesrc-build --run printenv
KDE_SESSION_VERSION=4
SDL_AUDIODRIVER=alsa
LANGUAGE=
XCURSOR_THEME=Oxygen_Blue
LESS=-R -M --shift 5
QMAIL_CONTROLDIR=/var/qmail/control
... etc.
```

## kdesrc-build Script Manual

### **--prefix=</path/to/kde>**

This allows you to change the directory that KDE will be installed to from the command line. This option implies `--reconfigure`, but using `--refresh-build` may still be required.

### **--revision**

This option causes kdesrc-build to checkout a specific numbered revision for each Subversion module, overriding any [branch](#), [tag](#), or [revision](#) options already set for these modules.

This option is likely not a good idea, and is only supported for compatibility with older scripts.

### **--build-system-only**

This option causes kdesrc-build to abort building a module just before the **make** command would have been run. This is supported for compatibility with older versions only, this effect is not helpful for the current KDE build system.

### **--install**

If this is the only command-line option, it tries to install all of the modules contained in `log/latest/build-status`. If command-line options are specified after `--install`, they are all assumed to be modules to install (even if they did not successfully build on the last run).

### **--no-snapshots**

Supplying this option causes kdesrc-build to always perform a normal initial checkout of a module instead of using a quick-start snapshot (only available for Git modules from the `kde-projects` repository). Note that this option should only be used if there is a failure using snapshots, as the quick-start snapshot reduces load on the KDE source repositories.

#### NOTE

Module snapshots *are* real checkouts. You should not need to specify this option, it is only a troubleshooting aid.

### **--delete-my-patches**

This option is used to let kdesrc-build delete source directories that may contain user data, so that the module can be re-downloaded. This would normally only be useful for KDE developers (who might have local changes that would be deleted).

This is currently only used to checkout modules that have been converted from Subversion to Git. You should not use this option normally, kdesrc-build will prompt to be re-run with it if it is needed.

### **--delete-my-settings**

This option is used to let kdesrc-build overwrite existing files which may contain user data.

This is currently only used for xsession setup for the login manager. You should not use this option normally, kdesrc-build will prompt to be re-run with it if it is needed.

### **--<option-name>=**

You can use this option to override an option in your [configuration file](#) for every module. For instance, to override the [log-dir](#) option, you would do: `--log-dir= /path/to/dir`.

#### NOTE

This feature can only be used for option names already recognized by kdesrc-build, that are not already supported by relevant command line options. For example the [async](#) configuration file option has specific `--async` and `--no-async` command line options that are preferred by kdesrc-build.



## kdesrc-build Script Manual

**`--set-module-option-value=<module-name>, <option-name>, <option-value>`**

You can use this option to override an option in your [configuration file](#) for a specific module.

Any other command-line options are assumed to be modules to update and build. Please, do not mix building with installing.

## Chapter 6

# Using kdesrc-build

### 6.1 Preface

Normally using `kdesrc-build` after you have gone through chapter 2 is as easy as doing the following from a terminal prompt:

```
% kdesrc-build
```

`kdesrc-build` will then download the sources for KDE, try to configure and build them, and then install them.

Read on to discover how `kdesrc-build` does this, and what else you can do with this tool.

### 6.2 Basic kdesrc-build features

#### 6.2.1 qt support

`kdesrc-build` supports building the Qt™ toolkit used by KDE software as a convenience to users. This support is handled by a special module named `qt`.

#### NOTE

Qt™ is developed under a separate repository from KDE software located at <http://code.qt.io/cgit/qt/>.

In order to build Qt™, you should make sure that the `qtdir` setting is set to the directory you'd like to install Qt™ to, as described in Section 2.2.

You should then ensure that the `qt` module is added to your `.kdesrc-buildrc`, before any other modules in the file. If you are using the sample configuration file, you can simply uncomment the existing `qt` module entry.

Now you should verify the `repository` option and `branch` options are set appropriately:

1. The first option is to build Qt™ using a mirror maintained on the KDE source repositories (no other changes are applied, it is simply a clone of the official source). This is highly recommended due to occasional issues with cloning the full Qt™ module from its official repository.

You can set the `repository` option for the `qt` module to `kde:qt` to use this option.

2. Otherwise, to build the standard Qt™, set your `repository` option to `git://gitorious.org/qt/qt.git`. Note that you may experience problems performing the initial clone of Qt™ from this repository.

In both cases, the `branch` option should be set to `master` (unless you'd like to build a different branch).

## 6.2.2 Standard flags added by kdesrc-build

Nota Bene: this section does not apply to modules for which you have configured a custom toolchain, using e.g. [cmake-toolchain](#).

To save you time, kdesrc-build adds some standard paths to your environment for you:

- The path to the KDE and Qt™ libraries is added to the `LD_LIBRARY_PATH` variable automatically. This means that you do not need to edit [libpath](#) to include them.
- The path to the KDE and Qt™ development support programs are added to the `PATH` variable automatically. This means that you do not need to edit [binpath](#) to include them.
- The path to the KDE-provided `pkg-config` is added automatically to `PKG_CONFIG_PATH`. This means that you do not need to use [set-env](#) to add these.
- The setting for [kdedir](#) is automatically propagated to the `KDEDIR` environment variable while building. (`KDEDIRS` is not affected).
- The setting for [qtdir](#) is automatically propagated to the `QTDIR` environment variable while building.

## 6.2.3 Changing kdesrc-build's build priority

Programs can run with different priority levels on Operating Systems, including Linux® and BSD. This allows the system to allocate time for the different programs in accordance with how important they are.

kdesrc-build will normally allocate itself a low priority so that the rest of the programs on your system are unaffected and can run normally. Using this technique, kdesrc-build will use extra CPU when it is available.

kdesrc-build will still maintain a high enough priority level so that it runs before routine batch processes and before CPU donation programs such as [Seti@Home](#).

To alter kdesrc-build so that it uses a higher (or lower) priority level permanently, then you need to adjust the [niceness](#) setting in the [configuration file](#). The [niceness](#) setting controls how 'nice' kdesrc-build is to other programs. In other words, having a higher [niceness](#) gives kdesrc-build a lower priority. So to give kdesrc-build a higher priority, reduce the [niceness](#) (and vice versa). The [niceness](#) can go from 0 (not nice at all, highest priority) to 20 (super nice, lowest priority).

You can also temporarily change the priority for kdesrc-build by using the [--nice command line option](#). The value to the option is used exactly the same as for [niceness](#).

### NOTE

It is possible for some programs run by the super user to have a negative nice value, with a correspondingly even higher priority for such programs. Setting a negative (or even 0) [niceness](#) for kdesrc-build is not a great idea, as it will not help run time significantly, but will make your computer seem very sluggish should you still need to use it.

## kdesrc-build Script Manual

To run `kdesrc-build` with a niceness of 15 (a lower priority than normal):

```
% kdesrc-build --nice=15
```

Or, you can edit the [configuration file](#) to make the change permanent:

```
niceness 15
```

### TIP

The [niceness](#) option only affects the usage of the computer's processor(s). One other major affect on computer performance relates to how much data input or output (I/O) a program uses. In order to control how much I/O a program can use, modern Linux<sup>®</sup> operating systems support a similar tool called `ionice`. `kdesrc-build` supports `ionice`, (but only to enable or disable it completely) using the [use-idle-io-priority](#) option, since `kdesrc-build` version 1.12.

## 6.2.4 Installation as the superuser

You may wish to have `kdesrc-build` run the installation with super user privileges. This may be for the unrecommended system-wide installation. This is also useful when using a recommended single user KDE build, however. This is because some modules (especially `kdebase`) install programs that will briefly need elevated permissions when run. They are not able to achieve these permission levels unless they are installed with the elevated permissions.

You could simply run `kdesrc-build` as the super user directly, but this is not recommended, since the program has not been audited for that kind of use. Although it should be safe to run the program in this fashion, it is better to avoid running as the super user when possible.

To take care of this, `kdesrc-build` provides the [make-install-prefix](#) option. You can use this option to specify a command to use to perform the installation as another user. The recommended way to use this command is with the `Sudo` program, which will run the `install` command as the super user.

For example, to install all modules using `Sudo`, you could do something like this:

```
global
  make-install-prefix sudo
  # Other options
end global
```

To use [make-install-prefix](#) for only a single module, this would work:

```
module svn-module-name
  make-install-prefix sudo
end module
```

## 6.2.5 Showing the progress of a module build

This feature is always available, and is automatically enabled when possible. What this does is display an estimated build progress while building a module; that way you know about how much longer it will take to build a module.

## 6.3 Advanced features

### 6.3.1 Partially building a module

It is possible to build only pieces from a single KDE module. For example, you may want to compile only one program from a module. `kdesrc-build` has features to make this easy. There are several complementing ways to do this.

#### 6.3.1.1 Checking out portions of a module

This is perhaps the best way to do this. When it works, it will save you download time and disk space. What happens is that `kdesrc-build` will download only the parts of a module that you specify. This is done using the `checkout-only` option for a module, which will specify a list of directories to download.

##### TIP

If you do not already know what to download from a module, it may be a good idea to browse the Subversion layout for a module first, using [WebSVN](#).

To only grab KUser and KSystemLog from `kdeadmin`, you could use `checkout-only` like this:

```
module kdeadmin
  checkout-only kuser ksystemlog
end module
```

##### IMPORTANT

The directories will be built in the order they are listed in the option. If one of the directories needs something else from the module to compile, then you need to make sure they are both in the `checkout-only` line, and that the required dependency goes before the directory that needs it.

Also, sometimes an application may need other directories and it is hard to figure out what they are, which may require some trial and error of constantly adding directories to the option to figure out. This option depends on support from the build system of the module, so it is only useful for modules that are collections of individual applications.

One final note to make about this option: If you change the value of this option, you should use `kdesrc-build --refresh-build module` in order to ensure that the module is reconfigured properly. In addition, `kdesrc-build` will never remove existing files if you take away the number of directories from your `checkout-only` option, or add the option to a module that has already been checked out.

#### 6.3.1.2 Removing directories from a build

Instead of restricting what is downloaded, it is possible to download everything but have the build system leave out a few directories when it does the build. This may be useful if one directory always breaks and is unnecessary to the rest of the module.

This is controlled with the `do-not-compile` option. It works similar to the `checkout-only` option just described, in that it is simply a list of directories that should not be compiled.

**IMPORTANT**

Also like [checkout-only](#), this option requires at least that the build system for the module is reconfigured after changing it. This is done using the `kdesrc-build --reconfigure module` command.

To remove the `python` directory from the `kdebindings` build process:

```
module kdebindings
  do-not-compile python
end module
```

**NOTE**

This function depends on some standard conventions used in most KDE modules. Therefore it may not work for all programs.

## 6.3.2 Branching and tagging support for `kdesrc-build`

### 6.3.2.1 What are branches and tags?

Subversion supports managing the history of the KDE source code. KDE uses this support to create branches for development, and to tag the repository every so often with a new version release.

For example, the KMail developers may be working on a new feature in a different branch in order to avoid breaking the version being used by most developers. This branch has development ongoing inside it, even while the main branch (called `/trunk`) may have development going on inside of it.

A tag, on the other hand, is a snapshot of the source code repository at a position in time. This is used by the KDE administration team to mark off a version of code suitable for release and still allow the developers to work on the code.

In Subversion, there is no difference between branches, tags, or trunk within the code. It is only a convention used by the developers. This makes it difficult to properly support branches and tags within `kdesrc-build`. However, there are some things that can be done.

### 6.3.2.2 How to use branches and tags

Support for branches and tags is handled by a set of options, which range from a generic request for a version, to a specific URL to download for advanced users.

The easiest method is to use the [branch](#) and [tag](#) options. You simply use the option along with the name of the desired branch or tag for a module, and `kdesrc-build` will try to determine the appropriate location within the KDE repository to download from. For most KDE modules this works very well.

To download `kdelibs` from KDE 4.6 (which is simply known as the 4.6 branch):

```
module kdelibs
  branch 4.6
  # other options...
end module
```

Or, to download `kdemultimedia` as it was released with KDE 4.6.1:

```
module kdemultimedia
  tag 4.6.1
  # other options...
end module
```

### TIP

You can specify a global branch value. But if you do so, do not forget to specify a different branch for modules that should not use the global branch!

### 6.3.2.3 Advanced branch support options

kdesrc-build supports two options for situations where `branch` and `tag` guess the correct path improperly: `module-base-path` and `override-url`.

- `module-base-path` is used to help kdesrc-build fill in the missing part of a module's path. In the KDE repository, all of the paths are of the form `svnRoot/module-base-path/ module-name`. Normally kdesrc-build can figure out the appropriate middle part by itself. When it cannot, you can use `module-base-path`, like this:

```
module kdesupport
  # kdesupport supports various tags to easily organize the required
  # software for a given KDE Platform release.
  module-base-path tags/kdesupport-for-4.5
end module
```

This would cause kdesrc-build to download kdesupport from (in this example), `svn://anonsvn.kde.org/home/kde/ tags/kdesupport-for-4.5`.

### TIP

In previous versions of kdesrc-build, the `module-base-path` was handled differently. If you encounter trouble using an old `module-base-path` definition perhaps you should verify that the actual path is as kdesrc-build expects by using the `--pretend` option.

- The `override-url` option, on the other hand, requires you to specify the exact path to download from. However, this allows you to pull from paths that previous versions of kdesrc-build would have no hope of downloading from. Currently, the `module-base-path` option should be sufficient for any Subversion source URL.

### IMPORTANT

kdesrc-build will not touch or correct the value you specify for `override-url` at all, so if you change your `svn-server` setting, you may need to update this as well.

## 6.3.3 How kdesrc-build tries to ensure a successful build

### 6.3.3.1 Automatic rebuilds

kdesrc-build used to include features to automatically attempt to rebuild the module after a failure (as sometimes this re-attempt would work, due to bugs in the build system at that time). Thanks to switching to CMake the build system no longer suffers from these bugs, and so kdesrc-build will not try to build a module more than once. There are situations where kdesrc-build will automatically take action though:

- If you change [configure-flags](#) or [cmake-options](#) for a module, then kdesrc-build will detect that and automatically re-run configure or cmake for that module.
- If the buildsystem does not exist (even if kdesrc-build did not delete it) then kdesrc-build will automatically re-create it. This is useful to allow for performing a full [--refresh-build](#) for a specific module without having that performed on other modules.

### 6.3.3.2 Manually rebuilding a module

If you make a change to a module's option settings, or the module's source code changes in a way kdesrc-build does not recognize, you may need to manually rebuild the module.

You can do this by simply running `kdesrc-build --refresh-build module`.

If you would like to have kdesrc-build automatically rebuild the module during the next normal build update instead, you can create a special file. Every module has a build directory. If you create a file called `.refresh-me` in the build directory for a module, kdesrc-build will rebuild the module next time the build process occurs, even if it would normally perform the faster incremental build.

#### TIP

By default, the build directory is `~/kdesrc/build/ module / .`. If you change the setting of the [build-dir](#) option, then use that instead of `~/kdesrc/build`.

Rebuild using `.refresh-me` for module `kdelibs`:

```
% touch ~/kdesrc/build/kdelibs/.refresh-me
% kdesrc-build
```

## 6.3.4 Changing environment variable settings

Normally kdesrc-build uses the environment that is present when starting up when running programs to perform updates and builds. This is useful for when you are running kdesrc-build from the command line.

However, you may want to change the setting for environment variables that kdesrc-build does not provide an option for directly. (For instance, to setup any required environment variables when running kdesrc-build on a timer such as Cron) This is possible with the [set-env](#) option.

Unlike most options, it can be set more than once, and it accepts two entries, separated by a space. The first one is the name of the environment variable to set, and the remainder of the line is the value.

Set `DISTRO=BSD` for all modules:

```
global
  set-env DISTRO BSD
end global
```

## 6.3.5 Resuming builds

### 6.3.5.1 Resuming a failed or canceled build

You can tell kdesrc-build to start building from a different module than it normally would. This can be useful when a set of modules failed, or if you canceled a build run in the middle. You can control this using the [--resume-from](#) option and the [--resume-after](#) option.



### NOTE

Older versions of kdesrc-build would skip the source update when resuming a build. This is no longer done by default, but you can always use the `--no-src` command line option to skip the source update.

Resuming the build starting from kdebase:

```
% kdesrc-build --resume-from=kdebase
```

Resuming the build starting after kdebase (in case you manually fixed the issue and installed the module yourself):

```
% kdesrc-build --resume-after=kdebase
```

If the last kdesrc-build build ended with a build failure, you can also use the `--resume` command line option, which resumes the last build starting at the module that failed. The source and metadata updates are skipped as well (but if you need these, it's generally better to use `--resume-from` instead).

### 6.3.5.2 Ignoring modules in a build

Similar to the way you can [resume the build from a module](#), you can instead choose to update and build everything normally, but ignore a set of modules.

You can do this using the `--ignore-modules` option. This option tells kdesrc-build to ignore all the modules on the command line when performing the update and build.

Ignoring extragear/multimedia and kdereview during a full run:

```
% kdesrc-build --ignore-modules extragear/multimedia kdereview
```

## 6.3.6 Changing options from the command line

### 6.3.6.1 Changing global options

You can change the setting of options read from the [configuration file](#) directly from the command line. This change will override the configuration file setting, but is only temporary. It only takes effect as long as it is still present on the command line.

kdesrc-build allows you to change options named like *option-name* by passing an argument on the command line in the form `--option-name=value`. kdesrc-build will recognize whether it does not know what the option is, and search for the name in its list of option names. If it does not recognize the name, it will warn you, otherwise it will remember the value you set it to and override any setting from the configuration file.

Setting the `source-dir` option to `/dev/null` for testing:

```
% kdesrc-build --pretend --source-dir=/dev/null
```

### 6.3.6.2 Changing module options

It is also possible to change options only for a specific module. The syntax is similar: `--module,option-name=value`.

This change overrides any duplicate setting for the module found in the [configuration file](#), and applies only while the option is passed on the command line.

Using a different build directory for the kdeedu module:

```
% kdesrc-build --kdeedu,build-dir=temp-build
```

## 6.4 Features for KDE developers

### 6.4.1 SSH Agent checks

kdesrc-build can ensure that KDE developers that use SSH to access the KDE source repository do not accidentally forget to leave the SSH Agent tool enabled. This can cause kdesrc-build to hang indefinitely waiting for the developer to type in their SSH password, so by default kdesrc-build will check if the Agent is running before performing source updates.

#### NOTE

This is only done for KDE developers using SSH. This is because no password is required for the default anonymous checkout. Subversion will handle passwords for the second possible protocol for KDE developers, https.

You may wish to disable the SSH Agent check, in case of situations where kdesrc-build is mis-detecting the presence of an agent. To disable the agent check, set the `disable-agent-check` option to **true**.

Disabling the SSH agent check:

```
global
  disable-agent-check true
end global
```

## 6.5 Other kdesrc-build features

### 6.5.1 Changing the amount of output from kdesrc-build

kdesrc-build has several options to control the amount of output the script generates. In any case, errors will always be output.

- The `--quiet` option (short form is `-q`) causes kdesrc-build to be mostly silent. Only important messages, warnings, or errors will be shown. When available, build progress information is still shown.
- The `--really-quiet` option (no short form) causes kdesrc-build to only display important warnings or errors while it is running.
- The `--verbose` option (short form is `-v`) causes kdesrc-build to be very detailed in its output.
- The `--debug` option is for debugging purposes only, it causes kdesrc-build to act as if `--verbose` was turned on, causes commands to also output to the terminal, and will display debugging information for many functions.

### 6.5.2 Color output

When being run from Konsole or a different terminal, kdesrc-build will normally display with colorized text.

You can disable this by using the `--no-color` on the command line, or by setting the `colorful-output` option in the [configuration file](#) to **false**.

Disabling color output in the configuration file:

```
global
  colorful-output false
end global
```

### 6.5.3 Removing unneeded directories after a build

Are you short on disk space but still want to run a bleeding-edge KDE checkout? `kdesrc-build` can help reduce your disk usage when building KDE from Subversion.

#### NOTE

Be aware that building KDE does take a lot of space. There are several major space-using pieces when using `kdesrc-build`:

1. The actual source checkout can take up a fair amount of space. The default modules take up about 1.6 gigabytes of on-disk space. You can reduce this amount by making sure that you are only building as many modules as you actually want. `kdesrc-build` will not delete source code from disk even if you delete the entry from the [configuration file](#), so make sure that you go and delete unused source checkouts from the source directory. Note that the source files are downloaded from the Internet, you *should not* delete them if you are actually using them, at least until you are done using `kdesrc-build`.

Also, if you already have a Qt™ installed by your distribution (and the odds are good that you do), you probably do not need to install the `qt` module. That will shave about 200 megabytes off of the on-disk source size.

One thing to note is that due to the way Subversion works: there are actually two files on disk for every file checked-out from the repository. `kdesrc-build` does not have code at this point to try and minimize the source size when the source is not being used.

2. `kdesrc-build` will create a separate build directory to build the source code in. Sometimes `kdesrc-build` will have to copy a source directory to create a fake build directory. When this happens, space-saving symlinks are used, so this should not be a hassle on disk space. The build directory will typically be much larger than the source directory for a module. For example, the build directory for `kdebase` is about 1050 megabytes, whereas `kdebase`'s source is only around 550 megabytes.

Luckily, the build directory is not required after a module has successfully been built and installed. `kdesrc-build` can automatically remove the build directory after installing a module, see the examples below for more information. Note that taking this step will make it impossible for `kdesrc-build` to perform the time-saving incremental builds.

3. Finally, there is disk space required for the actual installation of KDE, which does not run from the build directory. This typically takes less space than the build directory. It is harder to get exact figures however.

How do you reduce the space requirements of KDE? One way is to use the proper compiler flags, to optimize for space reduction instead of for speed. Another way, which can have a large effect, is to remove debugging information from your KDE build.

#### WARNING

You should be very sure you know what you are doing before deciding to remove debugging information. Running bleeding-edge software means you are running software which is potentially much more likely to crash than a stable release. If you are running software without debugging information, it can be very hard to create a good bug report to get your bug resolved, and you will likely have to re-enable debugging information for the affected application and rebuild to help a developer fix the crash. So, remove debugging information at your own risk!

Removing the build directory after installation of a module. The source directory is still kept, and debugging is enabled:

## kdesrc-build Script Manual

```
global
  configure-flags      --enable-debug
  remove-after-install builddir      # Remove build directory after  ↔
  install
end global
```

Removing the build directory after installation, without debugging information, with size optimization.

```
global
  cxxflags             -Os             # Optimize for size
  configure-flags      --disable-debug
  remove-after-install builddir      # Remove build directory after  ↔
  install
end global
```

## Chapter 7

# CMake, the KDE build system

### 7.1 Introduction to CMake

In March 2006, the CMake program beat out several competitors and was selected to be the build system for KDE 4, replacing the autotools-based system that KDE had used from the beginning.

A introduction to CMake page is available on the [KDE Community Wiki](#). Basically, instead of running `make -f Makefile.cvs`, then `configure`, then `Make`, we simply run CMake and then `Make`.

kdesrc-build has support for CMake. A few features of kdesrc-build were really features of the underlying buildsystem, including `configure-flags` and `do-not-compile`. When equivalent features are available, they are provided. For instance, the equivalent to the `configure-flags` option is `cmake-options`, and the `do-not-compile` option is also supported for CMake as of kdesrc-build version 1.6.3.

## Chapter 8

# Credits And License

This documentation is licensed under the terms of the [GNU Free Documentation License](#).

## Appendix A

# KDE modules and source code organization

### A.1 The ‘Module’

KDE groups its software into ‘modules’ of various size. This was initially a loose grouping of a few large modules, but with the introduction of the [Git-based source code repositories](#), these large modules were further split into many smaller modules.

kdesrc-build uses this module concept as well. In essence, a ‘module’ is a grouping of code that can be downloaded, built, tested, and installed.

#### A.1.1 Individual modules

It is easy to set kdesrc-build to build a single module. The following listing is an example of what a declaration for a Subversion-based module would look like in [the configuration file](#).

```
module kdefoo
  cmake-options -DCMAKE_BUILD_TYPE=Debug
end module
```

#### TIP

This is a Subversion-based module since it doesn’t use a [repository](#) option. Also, the `cmake-options` option is listed as an example only, it is not required.

#### A.1.2 Groups of related modules

Now most KDE source modules are Git-based KDE, and are normally combined into groups of modules.

kdesrc-build therefore supports groups of modules as well, using [module sets](#). An example:

```
module-set base-modules
  repository kde-projects
  use-modules kde-runtime kde-workspace kde-baseapps
end module-set
```

**TIP**

You can leave the module set name (*base-modules* in this case) empty if you like. This repository setting tells kdesrc-build where to download the source from, but you can also use a git:// URL.

One special feature of the ‘repository kde-projects’ is that kdesrc-build will automatically include any Git modules that are grouped under the modules you list (in the KDE Project database).

**A.1.3 Module ‘branch groups’**

Taking the concept of a [group of modules](#) further, the KDE developers eventually found that synchronizing the names of the Git branches across a large number of repositories was getting difficult, especially during the development push for the new KDE Frameworks for Qt™ 5.

So the concept of ‘branch groups’ was developed, to allow users and developers to select one of only a few groups, and allow the script to automatically select the appropriate Git branch.

kdesrc-build supports this feature as of version 1.16-pre2, via the [branch-group](#) option.

**Example A.1** Example of using branch-group

branch-group can be used in the configuration file as follows:

```
global
  # Select KDE Frameworks 5 and other Qt5-based apps
  branch-group kf5-qt5

  # Other global options here ...
end global

module-set
  # branch-group only works for kde-projects
  repository kde-projects

  # branch-group is inherited from the one set globally, but could
  # specified here.

  use-modules kdelibs kde-workspace
end module-set

# kdelibs's branch will be "frameworks"
# kde-workspace's branch will be "master" (as of August 2013)
```

In this case the same `branch-group` gives different branch names for each Git module.

This feature requires some data maintained by the KDE developers in a Git repository named `kde-build-metadata`, however this module will be included automatically by kdesrc-build (though you may see it appear in the script output).

**TIP**

KDE modules that do not have a set branch name for the branch group you choose will default to an appropriate branch name, as if you had not specified `branch-group` at all.



## Appendix B

# Superseded profile setup procedures

### B.1 Setting up a KDE login profile

These instructions cover how to setup the profile required to ensure your computer can login to your newly-built KDE Plasma desktop. `kdesrc-build` will normally try to do this automatically (see Section 2.5.1). This appendix section can be useful for those who cannot use `kdesrc-build`'s support for login profile setup. However the instructions may not always be up-to-date, it can also be useful to consult the `kde-env-master.sh` file included with the `kdesrc-build` source.

#### B.1.1 Changing your startup profile settings

##### IMPORTANT

The `.bash_profile` is the login settings file for the popular bash shell used by many Linux® distributions. If you use a different shell, then you may need to adjust the samples given in this section for your particular shell.

Open or create the `.bash_profile` file in the home directory with your favorite editor, and add to the end of the file: If you are building the qt module (you are by default), add instead:

```
QTDIR=(path to qtdir) # Such as ~/kdesrc/build/qt by default.
KDEDIR=(path to kdedir) # Such as ~/kde by default.
KDEDIRS=$KDEDIR
PATH=$KDEDIR/bin:$QTDIR/bin:$PATH
MANPATH=$QTDIR/doc/man:$MANPATH

# Act appropriately if LD_LIBRARY_PATH is not already set.
if [ -z $LD_LIBRARY_PATH ]; then
  LD_LIBRARY_PATH=$KDEDIR/lib:$QTDIR/lib
else
  LD_LIBRARY_PATH=$KDEDIR/lib:$QTDIR/lib:$LD_LIBRARY_PATH
fi

export QTDIR KDEDIRS PATH MANPATH LD_LIBRARY_PATH
```

or, if you are not building qt (and are using your system Qt™ instead), add this instead:

## kdesrc-build Script Manual

```
KDEDIR=(path to kdedir) # Such as ~/kde by default.
KDEDIRS=$KDEDIR
PATH=$KDEDIR/bin:$QTDIR/bin:$PATH

# Act appropriately if LD_LIBRARY_PATH is not already set.
if [ -z $LD_LIBRARY_PATH ]; then
  LD_LIBRARY_PATH=$KDEDIR/lib
else
  LD_LIBRARY_PATH=$KDEDIR/lib:$LD_LIBRARY_PATH
fi

export KDEDIRS PATH LD_LIBRARY_PATH
```

If you are not using a dedicated user, set a different \$KDEHOME for your new environment in your `.bash_profile`:

```
export KDEHOME="${HOME}/.kde-svn"

# Create it if needed
[ ! -e ~/.kde-svn ] && mkdir ~/.kde-svn
```

### NOTE

If later your K Menu is empty or too crowded with applications from your distribution, you may have to set the XDG environment variables in your `.bash_profile`:

```
XDG_CONFIG_DIRS="/etc/xdg"
XDG_DATA_DIRS="${KDEDIR}/share:/usr/share"
export XDG_CONFIG_DIRS XDG_DATA_DIRS
```

## B.1.2 Starting KDE

Now that you have adjusted your environment settings to use the correct KDE, it is important to ensure that the correct `startkde` script is used as well.

Open the `.xinitrc` text file from the home directory, or create it if necessary. Add the line:

```
exec ${KDEDIR}/bin/startkde
```

### IMPORTANT

On some distributions, it may be necessary to perform the same steps with the `.xsession` file, also in the home directory. This is especially true when using graphical login managers such as `kdm`, `gdm`, or `xdm`.

Now start your fresh KDE: in BSD and Linux<sup>®</sup> systems with virtual terminal support, **Ctrl+Alt+F1 ... Ctrl+Alt+F12** keystroke combinations are used to switch to Virtual Console 1 through 12. This allows you to run more than one desktop environment at the same time. The first six are text terminals and the following six are graphical displays.

If when you start your computer you are presented to the graphical display manager instead, you can use the new KDE environment, even if it is not listed as an option. Most display managers, including `kdm`, have an option to use a 'Custom Session' when you login. With this option, your

## kdesrc-build Script Manual

session settings are loaded from the `.xsession` file in your home directory. If you have already modified this file as described above, this option should load you into your new KDE installation.

If it does not, there is something else you can try that should normally work: Press **Ctrl+Alt+F2**, and you will be presented to a text terminal. Log in using the dedicated user and type:

```
startx -- :1
```

### TIP

You can run the KDE from sources and the old KDE at the same time! Log in using your regular user, start the stable KDE desktop. Press **Ctrl+Alt+F2** (or **F1**, **F3**, etc..), and you will be presented with a text terminal. Log in using the dedicated KDE Subversion user and type:

```
startx -- :1
```

You can go back to the KDE desktop of your regular user by pressing the shortcut key for the already running desktop. This is normally **Ctrl+Alt+F7**, you may need to use **F6** or **F8** instead. To return to your kdesrc-build-compiled KDE, you would use the same sequence, except with the next function key. For example, if you needed to enter **Ctrl+Alt+F7** to switch to your regular KDE, you would need to enter **Ctrl+Alt+F8** to go back to your kdesrc-build KDE.