kdesrc-build Script Manual

Michael Pyne
Carlos Woelz

Contents

List of Tables

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 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:

- 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 TechBase 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 to control which modules are built, where they are installed to, etc. This file is located at `~/.config/kdesrc-buildrc` (`$XDG_CONFIG_HOME/kdesrc-buildrc`, if `$XDG_CONFIG_HOME` is set).

You can use kdesrc-build --generate-config in order to prepare a simple kdesrc-build configuration. You can then edit the `~/.config/kdesrc-buildrc` configuration file to make any changes you see fit.

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 `~/.config/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. 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 `~/.config` (or in `$XDG_CONFIG_HOME`, if set) directory called `kdesrc-buildrc`, which sets the general options and specifies 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 that the configuration is stored in `~/.config/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:

- install-dir, which changes the destination directory that your KDE software is installed to. This defaults to `~/kde/usr`, 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 `~/kde/src`.
- 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
```

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
```

This command will download the appropriate source code, build and install each module in order. 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 `~/kdesrc/log`. To see what the 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 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.

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 sddm 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® distribution requires the `xorg-x11-xinit-session` package to be installed for custom `xsession` login support.
- Debian and Debian-derived Linux® 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)
```

   ❶ 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 `kde-env-master.sh.in` 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 descendents 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

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 allows you to checkout modules quickly. If the module you are checking out has already been checked out previously, then kdesrc-build will download only commits that are not yet 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 Git 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 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 git+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:

```
install-dir ~/kde/usr
```

- 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 `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.

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 `git-checkout-update.log` (for a module checkout or 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 Git 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 Git modules, *module-name* must be a module from the KDE Git repository (for example, kdeartwork or kde-wallpapers).

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.

You can also introduce your own non-standard global variables for referencing them further in the config. To do this, your option name should be prepended with underscore symbol. Example:

Example 4.1 Introducing your own global option for referencing later in config

```
global
  _ver 6  # &#8592; your custom variable (starting with underscore)
  _kde ~/kde${_ver}  # &#8592; custom variable can contain another defined  ↩
      variable
  source-dir ${_kde}/src  # &#8592; note that nested variable (_kde &#8594; ↩
       _ver) is also resolved
end global

options kdepim
  log-dir /custom/path/logs${_ver} # &#8592; you can use custom variable  ↩
      just like a standard
end options
```

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 4.2 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.

You can use variables in the value of include instruction:

```
global
  _ver 6
  source-dir ~/kde${_ver}/src
  ...
  persistent-data-file ~/kde${_ver}/persistent-options.json
end global

include ~/kde6/src/kdesrc-build/data/build-include/kf${_ver}-qt${_ver}.ksb
```

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 `master`.
- configure-flags to define what flags to configure Qt™ with.
- install-dir, to set the directory to install KDE to.
- make-options, to pass options to the Make program (such as number of CPUs to use).
- qt-install-dir, to set the directory to install Qt™ to.
- source-dir, to change where to download the source code to.

4.2 Table of available configuration options

Here are tables of various options, containing the following information:

- The option name
- The scope of the option: *global*, *module* or *module-set*. Options in *module* or/and *module-set* scope can also be defined in *options* sections.
- Special comments on the purpose and usage of the option.

Option name	Description
async	Type: Boolean Default value: True Available since: 1.6 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. Related command-line option: `--async`

colorful-output	Type Default value Set this option to **`false`** to disable the colorful output of kdesrc-build. Note that kdesrc-build will not output the color codes to anything but a terminal (such as xterm, Konsole, or the normal Linux® console). Related command-line option: `--color` (or `--colorful-output`), `--no-color` (or `--no-colorful-output`)	Boolean True
disable-agent-check	Type Default value If you are using SSH to download the Git sources (such as if you are using the git+ssh protocol), this option controls if 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. Related command-line option: --disable-agent-check, --no-disable-agent-check	Boolean False

git-desired-protocol	Type Default value History information This option only applies to modules from a KDE project repository. What this option actually does is configure which network protocol to prefer when pushing source code for these modules. Normally the very-efficient `git` 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 `https` protocol used for Internet web sites. If you are using one of these constrained networks you can set this option to **`http`** to prefer `https` communications instead. TIP You may also need the http-proxy option if an HTTP proxy is also needed for network traffic. In any other situation you should not set this option as the default protocol is most efficient.	String git This option was a or to 20.06 this op fetch URL instea `https` is always u cts.

git-repository-base	Type String
	Available since 1.12.1
	This option 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 Git server mo`
	`  git-repository-base kde-git kde:`
	`end global`
	`# Module declarations`
	`module-set`
	`  # Now you can use the alias you defined earlier, but onl`
	`  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 aliases as necessary.
	TIP
	It is not required to use this option to take advantage of module-set, this option exists to make it easy to use the same repository across many different module sets.

install-environment-driver	Type Default value Available since 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: • `$XDG_CONFIG_HOME/kde-env-master.sh` (normally found at `~/.config/kde-env-master.sh`). • `$XDG_CONFIG_HOME/kde-env-user.sh` (normally found at `~/.config/kde-env-user.sh`). The `kde-env-user.sh` is optional. It is intended for user customizations (see the Troubleshooting and Debugging 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 *`false`*, and ensuring that the install-session-driver option is also disabled. TIP kdesrc-build will not overwrite your existing files (if present) unless you also pass the `--delete-my-settings` command-line option. Related command-line option: --install-environment-driver, --no-install-environment-driver	Boolean True 17.08

install-session-driver	Type Default value Available since 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: • `~/.xsession` • `$XDG_CONFIG_HOME/kde-env-master.sh` (normally found at `~/.config/kde-env-master.sh`). • `$XDG_CONFIG_HOME/kde-env-user.sh` (normally found at `~/.config/kde-env-user.sh`). If you maintain your own login driver then you can disable this feature by setting this option to *false*. If enabled, this feature also enables the install-environment-driver feature. TIP kdesrc-build will not overwrite your existing files (if present) unless you also pass the `--delete-my-settings` command-line option. Related command-line option: --install-session-driver, --no-install-session-driver	Boolean True 1.16
niceness	Type Default value 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. Related command-line option: `--nice` (or `--niceness`) *value*	Integer 10

num-cores	Type Default value Available since This option is defined by kdesrc-build (when using **kdesrc-build --generate-config**), 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. Related command-line option: --num-cores *value*	Integer Depends on syste 20.07
num-cores-low-mem	Type Default value Available since This option is defined by kdesrc-build (when using **kdesrc-build --generate-config**), 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. Related command-line option: --num-cores-low-mem *value*	Integer Depends on syste 20.07

persistent-data-file	Type Available since Use this option to change where kdesrc-build stores its persistent data. The default is to store this data in a file called `.kdesrc-build-data`, placed in the same directory as the configuration file in use. If the global configuration file is in use, it will be saved to `~/.local/state/kdesrc-build-data` (`$XDG_STATE_HOME/kdesrc-build-data`, if `$XDG_STATE_HOME` is set). If you have multiple available configurations in the same directory, you may want to manually set this option, so that different configurations do not end up with conflicting persistent data. Related command-line option: --persistent-data-file *file*	String 1.15
ssh-identity-file	Type Available since Set this option to control which private SSH key file is passed to the **ssh-add** command when kdesrc-build is downloading source code from repositories that require authentication. See also: Section 6.4.1.	String 1.14.2
use-idle-io-priority	Type Default value Available since Use lower priority for disk and other I/O, which can significantly improve the responsiveness of the rest of the system at the expense of slightly longer running times for kdesrc-build. Related command-line option: --use-idle-io-priority, --no-use-idle-io-priority	Boolean False 1.12
use-inactive-modules	Type Default value Allow kdesrc-build to also clone and pull from repositories marked as inactive. Related command-line option: --use-inactive-modules, --no-use-inactive-modules	Boolean False

Table 4.1: Global scope only options

Option name	**Description**

binpath	Type 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 `$PATH` that is set when the script starts. This environment variable should include the colon-separated paths of your development toolchain. The paths `${install-dir}/bin` and `${qt-install-dir}/bin` are automatically added. You may use the tilde (`~`) for any paths you add using this option. Related command-line option: --binpath *`path`*	String
branch	Type Default value Checkout the specified branch instead of the default branch. NOTE For most KDE modules you probably wish to use the branch-group option instead and use this option for case-by-case exceptions. Related command-line option: --branch *`value`*	String master

branch-group	Type Available since 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 `kde-build-metadata` source repository in your source directory). After a branch is determined that branch is used as if you had specified it yourself using the branch option. 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. Note that if you *do* choose a `branch` yourself, that it will override this setting. The same is true of other specific branch selection options such as tag. NOTE This option only applies to `kde-projects` Git modules (the common case). See also Section 2.6.4. Related command-line option: --branch-group *value*	String 1.16-pre2

build-dir	Type Default value Use this option to change the directory to contain the built sources. There are three different ways to use it: 1. Relative to the KDE Git source directory (see the source-dir option). This is the default, and is selected if you type a directory name that does not start with a tilde (~) or a slash (/). 2. Absolute path. If you specify a path that begins with a /, then that path is used directly. For example, `/tmp/kde-obj-dir/ .` 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`. Perhaps surprisingly, this option can be changed per module. Related command-line option: --build-dir *path*	String `~/kde/build`

build-when-unchanged	Type Default value Control whether kdesrc-build always tries to build a module that has not had any source code updates. If set to **`true`**, kdesrc-build always attempts the build phase for a module, even if the module did not have any source code updates. With this value it will more likely lead to a correct build. If set to **`false`**, 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. IMPORTANT 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. Related command-line option: `--build-when-unchanged` (or `--force-build`), `--no-build-when-unchanged` (or `--no-force-build`)	Boolean True
cmake-generator	Type Default value 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. Note that if a valid generator is also specified through cmake-options it will override the value for `cmake-generator`. Related command-line option: --cmake-generator *`value`*	String Unix Makefiles

cmake-toolchain	Type Specify a toolchain file to use with CMake. When a valid toolchain file is configured, kdesrc-build will *no longer set environment variables automatically*. You can use set-env, binpath and libpath to fix up the environment manually if your toolchain file does not work out of the box with kdesrc-build. Refer to the overview of standard flags added by kdesrc-build for more information. Note that if a valid toolchain is also specified through cmake-options it will override the value for `cmake-toolchain`. Related command-line option: --cmake-toolchain *`value`*	String

cmake-options	Type Appends to global options for the default buildsystem, overrides global for other buildsystems. 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. This option does not apply to qt (which does not use CMake). Use configure-flags instead. If a valid generator is specified among the listed options it will override the value of cmake-generator. Invalid (unsupported) generators are ignored and will not be passed to CMake. If a valid toolchain file is specified among the listed options it will override the value of cmake-toolchain. Invalid toolchains are ignored and will not be passed to CMake. Since these options are passed directly to the CMake command line, they should be given as they would be typed into CMake. For example: `cmake-options -DCMAKE_BUILD_TYPE=RelWithDebInfo` 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, *required* CMake parameters are set for you automatically) Related command-line option: --cmake-options *value*	String
compile-commands-export	Type Default value Enables the generation of a `compile_commands.json` via CMake inside the build directory. Related command-line option: --compile-commands-export, --no-compile-commands-export	Boolean True

compile-commands-linking	Type Default value Enables the creation of symbolic links from `compile_commands.json` generated via CMake inside the build directory to the matching source directory. Related command-line option: --compile-commands-linking, --no-compile-commands-linking	Boolean False
configure-flags	Type Appends to global options for the default buildsystem, overrides global for other buildsystems. Use this option to specify what flags to pass to ./configure 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. *This option only works for qt.* To change configuration settings for KDE modules, see cmake-options. Related command-line option: --configure-flags *value*	String
custom-build-command	Type This option can be set to run a different command (other than **make**, 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 make-options option as normal. Related command-line option: --custom-build-command *value*	String

cxxflags	Type Appends to global options for the default buildsystem, overrides global for other buildsystems. Use this option to specify what flags to use for building the module. This option is specified here instead of with configure-flags or cmake-options because this option will also set the environment variable `CXXFLAGS` during the build process. Note that for KDE 4 and any other modules that use CMake, it is necessary to set the CMAKE_BUILD_TYPE option to **none** when configuring the module. This can be done using the cmake-options option. Related command-line option: --cxxflags *value*	String
dest-dir	Type Use this option to change the name a module is given on disk. For example, if your module was extragear/network, you could rename it to extragear-network 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 build-dir or source-dir options. Related command-line option: --dest-dir *path*	String
do-not-compile	Type 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. For example, to disable building the `codeeditor` and `minimaltest` directories of the `syntaxhighlighting` framework, you would add **do-not-compile codeeditor minimaltest** compiling, you would add "do-not-compile juk kscd" to your syntaxhighlighting options. See Section 6.3.1.1 for an example. Related command-line option: --do-not-compile *value*	String

git-user	Type Available since 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 *newly downloaded* 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 `User Name <email@example.com>`. For instance, a developer named 'Foo Barbaz' with the email address 'foo@abc.xyz' would use: `git-user Foo Barbaz <foo@abc.xyz>`	String 15.09
http-proxy	Type Available since This option, if set, uses the specified URL as a proxy server to use for any HTTP network communications (for example, when downloading the KDE project database). In addition, kdesrc-build will try to ensure that the tools it depends on also use that proxy server, if possible, by setting the `http_proxy` environment variable to the indicated server, *if that environment variable is not already set*. Related command-line option: --http-proxy *value*	String 1.16

directory-layout	Type Valid values Default value This option is used to configure the layout which kdesrc-build should use when creating source and build directories. The **flat** 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 **metadata** layout would be `source/ktp-text-ui` using the **flat** layout instead. The **invent** layout creates a directory hierarchy mirroring the relative paths of repositories on invent.kde.org. For example `source/kde/applications/kate` in the **metadata** layout would be `source/utilities/kate` using the **invent** layout instead. This layout only affects KDE projects. It is a good choice for people starting out with kdesrc-build. Finally, the **metadata** 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. Related command-line option: --directory-layout *value*	String **flat**, **invent**, **m** flat

generate-vscode-project-config	Type Default value Module setting overrides global Set this option to **`true`** to make kdesrc-build create VS Code project files (.vscode directory) in the module source directory. The .vscode folder will be created in the project source directory, only if it does not already exist. The configurations will enable the use of LSP, building, debugging, and running the project from within VS Code. The configuration also recommends extensions to install that are useful for working on most KDE projects. Related command-line option: `--generate-vscode-project-config`, `--no-generate-vscode-project-config`	Boolean False
include-dependencies	Type Default value Controls if kdesrc-build will include known dependencies of this module in its build, without requiring you to mention those dependencies (even indirectly). NOTE This option only works for `kde-project`-based modules, and requires that the metadata maintained by the KDE developers is accurate for your selected branch-group. This is to support building applications that need versions of Qt™ or Plasma more recent than supported on common operating systems. Related command-line option: `--include-dependencies` (or `-d`), `--no-include-dependencies` (or `-D`)	Boolean True
install-after-build	Type Default value This option is used to install the package after it successfully builds. You can also use the `--no-install` command line flag. Related command-line option: --install-after-build, --no-install-after-build	String True

install-dir	Type Default value This option controls where to install the module after it is built. If you change this to a directory needing root access, you may want to read about the make-install-prefix option as well. Changing this option for specific module allows you to install it 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 `${MODULE}` or `$MODULE` in the path to have them expanded to the module's name. Related command-line option: `--install-dir path`	String `~/kde/usr`
libname	Type Default value Set this option to change the default name of the installed library directory inside ${install-dir} and ${qt-install-dir}. On many systems this is either "lib" or "lib64". Auto-detection is attempted to set the correct name by default, but if the guess is wrong then it can be changed with this setting. Related command-line option: --libname *value*	String Auto detected
libpath	Type Set this option to set the environment variable `LD_LIBRARY_PATH` while building. You cannot override this setting in a module option. The default value is blank, but the paths `${install-dir}/$LIBNAME` and `${qt-install-dir}/$LIBNAME` are automatically added. You may use the tilde (~) for any paths you add using this option. Related command-line option: --libpath *path*	String
log-dir	Type Use this option to change the directory used to hold the log files generated by the script. Related command-line option: --log-dir *path*	String

make-install-prefix	Type Set this variable to a space-separated list, which is interpreted as a command and its options to precede the **`make install`** command used to install modules. This is useful for installing packages with Sudo for example, but please be careful while dealing with root privileges. Related command-line option: --make-install-prefix *value*	String
make-options	Type Set this variable in order to pass command line options to the **make** command. This is useful for programs such as distcc or systems with more than one processor core. Note that not all supported build systems use **make**. For build systems that use **ninja** for build (such as the Meson build system), see the ninja-options setting. Related command-line option: --make-options *value*	String
manual-build	Type Default value Set the option value to **`true`** to keep the build process from attempting to build this module. It will still be kept up-to-date when updating from Git. This option is exactly equivalent to the `--no-build` command line option.	Boolean False
manual-update	Type Default value Set the option value to **`true`** 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.	Boolean False

ninja-options	Type
	Set this variable in order to pass command line options to the **ninja** build command. This can be useful to enable 'verbose' output or to manually reduce the number of parallel build jobs that **ninja** would use.
	
	NOTE
	Note that this setting only controls ninja when used by kdesrc-build. The Qt™ 'webengine' module uses **ninja** indirectly, but only officially supports being built by **make**. In this situation, you can set `NINJAFLAGS` as a way to have **make** pass the appropriate flags when it later calls **ninja**, by using make-options.
	
	`options qtwebengine`
	`  # Restrict make and ninja to using no more than 6 separ`
	`  # when more CPU is available, to avoid running out of m`
	`  make-options -j6 NINJAFLAGS=-j6`
	`end options`
	
	Related command-line option:
	--ninja-options *value*
		String

override-build-system	Type Default value Valid values Available since 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: **KDE** 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. **Qt** Used to build the Qt™ library itself. **qmake** Used to build Qt™ modules that use qmake-style `.pro` files. **generic** Used to build modules that use plain Makefiles and that do not require any special configuration. **autotools** This is the standard configuration tool used for most Free and open-source software not in any of the other categories. **meson** This is a relatively new tool gaining popularity as a replacement for the autotools and may be required for some non-KDE modules. Related command-line option: --override-build-system *value*	String Auto detected KDE, Qt, qmake, 1.16

purge-old-logs	Type Default value This option controls whether old log directories are automatically deleted or not. Related command-line option: --purge-old-logs, --no-purge-old-logs	Boolean True
qmake-options	Type Available since Any options specified here are passed to the **qmake** command, for modules that use the qmake build system. For instance, you can use the **`PREFIX=/path/to/qt`** option to qmake to override where it installs the module. Related command-line option: --qmake-options *`value`*	String 1.16
qt-install-dir	Type This option controls where to install qt modules after build. If you do not specify this option, kdesrc-build will assume that Qt™ is provided by the operating system. Related command-line option: --qt-install-dir *`path`*	String
remove-after-install	Type Valid values Default value 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. Possible values for this option are: • none - Do not delete anything. • builddir - Delete the build directory, but not the source. • all - Delete both the source code and build directory. Note that using this option can have a significant detrimental impact on both your bandwidth usage (if you use *`all`*) and the time taken to compile KDE software, since kdesrc-build will be unable to perform incremental builds. Related command-line option: --remove-after-install *`value`*	String none, builddir, al none

repository	Type Available since This option 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.	String 1.10
revision	Type Available since 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 branch 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. Related command-line option: `--revision id`	String 1.16
run-tests	Type Default value If set to **`true`**, 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. Related command-line option: --run-tests, --no-run-tests	Boolean False
set-env	Type 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 `RONALD` to McDonald, you would put in the appropriate section this command: `set-env RONALD McDonald` This option is special in that it can be repeated without overriding earlier set-env settings in the same section of the configuration file. This way you can set more than one environment variable per module (or globally).	String

source-dir	Type Default value This option is used to set the directory on your computer to store the KDE Git sources at. You may use the tilde (~) to represent the home directory if using this option. Related command-line option: --source-dir *path*	String ~/kde/src
stop-on-failure	Type Default value Setting this option to **false** allows the script to continue execution after an error occurs during the build or install process. Related command-line option: --stop-on-failure, --no-stop-on-failure	Boolean True
tag	Type Available since Use this option to download a specific release of a module. *Note:* The odds are very good that you *do not want* to use this option. KDE releases are available in tarball form from the KDE download site. Related command-line option: --tag *value*	String 1.16
use-clean-install	Type Default value Available since Set this option to **true** in order to have kdesrc-build run **make uninstall** directly before running **make install**. 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 **make uninstall**. Related command-line option: --use-clean-install, --no-use-clean-install	Boolean False 1.12

Table 4.2: All scopes (module, module-set and global) options

These options do not require any value (except ″filter-out-phases″). They are applied if they are presented in a section.

Option name	Scope
no-src	global module module-set

no-install	global module module-set
no-tests	global module module-set
no-build	global module module-set
build-only	global module module-set
install-only	global module module-set
uninstall	global module module-set
filter-out-phases	global module module-set

Table 4.3: Phase selection options

Option name	**Scope**
ignore-modules	global module-set
use-modules	module-set

Table 4.4: Modules selection 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

5.3.1 Generic

`--pretend` (or `--dry-run` 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.

--include-dependencies (or -d), --no-include-dependencies (or -D)

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 configuration file option is include-dependencies.

This option is enabled by default.

--ignore-modules (or -!) ***module [module ...]***

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).

Note that this option does not override ignore-modules config option in global section. Instead, it appends it.

--run (or --start-program) [-e|--exec name] [-f|--fork] ***program [parameters ...]***

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, source the prefix.sh to apply environment variables, and then execute the given program.

--revision ***id***

This option causes kdesrc-build to checkout a specific numbered revision for each Git 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.

--delete-my-patches, --no-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).

You should not use this option normally, kdesrc-build will prompt to be re-run with it if it is needed.

--delete-my-settings, --no-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> ***value***

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.

--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.

5.3.2 Resuming and stopping

`--resume-from` (or `--from` or `-f`) *`module`*

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

> NOTE
> 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 6.3.6.1. 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` (or `--after` or `-a`) *`module`*

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

> NOTE
> 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 6.3.6.1. 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`.

`--stop-before` (or `--until`) *`module`*

This 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`.

`--stop-after` (or `--to`) *`module`*

This 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`.

`--stop-on-failure`, `--no-stop-on-failure`

This option controls if the build will be aborted as soon as a failure occurs. Default behavior is --stop-on-failure. You may override it if you wish 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.

See also the stop-on-failure configuration file option.

--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.

5.3.3 Modules information

--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 'mode', which may 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.
- *module-set*, which causes kdesrc-build to output the name of module-set which contains the module. This can be used to generate zsh autocompletion cache.
- *build-system*, which causes kdesrc-build to output the name of build system detected for the module. This can be used to debug build system auto-detection problems, or when developing tests for specific build systems.
- Any option name that is valid for modules in the configuration file.

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

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

--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 the dependency would be built. Note: the generated output may become quite large for applications with many dependencies.

--dependency-tree-fullpath

Prints out dependency information on the modules that would be built using a tree format (recursive). In fullpath format. Note: the generated output may become quite large for applications with many dependencies.

--list-installed

Print installed modules and exit. This can be used to generate autocompletion for the --run option.

5.3.4 Exclude specific action

--no-metadata (or -M)

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.

--no-src (or -S)

Skip contacting the Git server.

--no-build

Skip the build process.

--no-install

Do not automatically install packages after they are built.

5.3.5 Only specific action

--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.

--src-only (or -s)

Only perform the source update.

--build-only

Only perform the build process.

--install-only

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

--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.

5.3.6 Build behavior

--build-when-unchanged (or --force-build), --no-build-when-unchanged (or --no-force-build)

Enabling 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. This option is enabled by default.

--refresh-build (or -r)

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.

--install-dir path

This allows you to change the directory where modules will be installed to. This option implies `--reconfigure`, but using `--refresh-build` may still be required.

--generate-vscode-project-config, --no-generate-vscode-project-config

Generate a `.vscode` directory with configurations for building and debugging in Visual Studio Code. This option is disabled by default.

5.3.7 Script runtime

--async, --no-async

Enables or disables the asynchronous mode, which can perform the source code updates and module builds at the same time. If disabled, the update will be performed in its entirety before the build starts. Disabling this option will slow down the overall process. If you encounter IPC errors while running kdesrc-build try disabling it, and submitting a bug report. This option is enabled by default.

--color (or --colorful-output), --no-color (or --no-colorful-output)

Enable or disable colorful output. By default, this option is enabled for interactive terminals.

--nice (or --niceness) *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 the lowest priority. This option defaults to 10.

--rc-file *file*

The file to read the configuration options from. The default value for this parameter is `kdesrc-buildrc` (checked in the current working directory). If this file doesn't exist, `~/.config/kdesrc-buildrc` (`$XDG_CONFIG_HOME/kdesrc-buildrc`, if `$XDG_CONFIG_HOME` is set) will be used instead. See also chapter 4.

5.3.8 Setup

--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® distributions)
- Adding required environment variables to `~/.bashrc`

This option is exactly equivalent to using `--install-distro-packages --generate-config` at the same time. In kdesrc-build (perl implementation) it additionally uses "--install-distro-packages-perl".

--install-distro-packages

Installs distro packages (on supported Linux® distributions) necessary to prepare the system for kdesrc-build to operate, and for the newly-installed KDE software to run.

See also `--initial-setup`

--generate-config

Generate the kdesrc-build configuration file.

See also `--initial-setup`

5.3.9 Verbosity level

--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.

--quiet (or --quite 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.

5.3.10 Script information

--version (or -v)

Display the program version.

--help (or -h)

Only display simple help on this script.

--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.

--show-options-specifiers

Print the specifier lines (in the format that GetOpts::Long accepts) for all command line options supported by the script. This may be used by developers, for example, for generating zsh autocompletion functions.

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 qt-install-dir option 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.

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.

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® 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 some-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 Removing directories from a build

It is possible to download an entire repository but have the build system leave out a few directories when it does the build. This requires that the module uses CMake and that the module's build system allows the directory to remove to be optional.

This is controlled with the do-not-compile option.

> IMPORTANT
> 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?

Git 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 master) may have development going on inside of it.

A tag, on the other hand, is a specified point in 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.

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!

6.3.3 Stopping the build early

6.3.3.1 The build normally continues even if failures occur

kdesrc-build normally will update, build and install all modules in the specified list of modules to build, even if a module fails to build. This is usually a convenience to allow you to update software packages even if a simple mistake is made in one of the source repositories during development that causes the build to break.

However you may wish for kdesrc-build to stop what it is doing once a module fails to build and install. This can help save you time that will be wasted trying to make progress when modules remaining in the build list will not be able to successfully build either, especially if you have not ever successfully built the modules in the list.

6.3.3.2 Not stopping early with --no-stop-on-failure

The primary method to do this is to use the --no-stop-on-failure command line option when you run kdesrc-build.

This option can also be set in the configuration file to make it the normal mode of operation.

It is also possible to tell kdesrc-build at runtime to stop building *after* completing the current module it is working on. This is as opposed to interrupting kdesrc-build using a command like **Ctrl**+**C**, which interrupts kdesrc-build immediately, losing the progress of the current module.

> IMPORTANT
> Interrupting kdesrc-build during a module install when the use-clean-install option is enabled will mean that the interrupted module will be unavailable until kdesrc-build is able to successfully build the module!
> If you need to interrupt kdesrc-build without permitting a graceful shutdown in this situation, at least try to avoid doing this while kdesrc-build is installing a module.

6.3.3.3 Stopping kdesrc-build gracefully when stop-on-failure is false

As mentioned above, it is possible to cause kdesrc-build to gracefully shutdown early once it has completed the module it is currently working on. To do this, you need to send the POSIX `HUP` signal to kdesrc-build

You can do this with a command such as **pkill** (on Linux® systems) as follows:

```
$ pkill -HUP kdesrc-build
```

If done successfully, you will see a message in the kdesrc-build output similar to:

```
[ build ] recv SIGHUP, will end after this module
```

> **NOTE**
> kdesrc-build may show this message multiple times depending on the number of individual kdesrc-build processes that are active. This is normal and not an indication of an error.

Once kdesrc-build has acknowledged the signal, it will stop processing after the current module is built and installed. If kdesrc-build is still updating source code when the request is received, kdesrc-build will stop after the module source code update is complete. Once both the update and build processes have stopped early, kdesrc-build will print its partial results and exit.

6.3.4 How kdesrc-build tries to ensure a successful build

6.3.4.1 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.

6.3.4.2 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 `~/kde/build/ module / .` If you change the setting of the build-dir option, then use that instead of `~/kde/build`.

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

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

6.3.5 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
```

6.3.6 Resuming builds

6.3.6.1 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).

6.3.6.2 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
```

6.3.7 Changing options from the command line

6.3.7.1 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
```

6.3.7.2 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.

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.

Debug level	Level name	Command line option
0	DEBUG	--debug
1	WHISPER	--verbose
2	INFO	Selected by default
3	NOTE	--quiet
4	WARNING	--really-quiet
5	ERROR	No way to select

Table 6.1: Table of debug levels

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 Git.

> 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.

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:

```
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

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 Git-based module would look like in the configuration file.

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

> TIP
> This is a Git-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.in` 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:

```
PATH=${install-dir}/bin:${qt-install-dir}/bin:$PATH
MANPATH=${qt-install-dir}/doc/man:$MANPATH

# Act appropriately if LD_LIBRARY_PATH is not already set.
if [ -z $LD_LIBRARY_PATH ]; then
  LD_LIBRARY_PATH=${install-dir}:/lib:${qt-install-dir}/lib
else
  LD_LIBRARY_PATH=${install-dir}:/lib:${qt-install-dir}/lib: ↩
      $LD_LIBRARY_PATH
fi

export PATH MANPATH LD_LIBRARY_PATH
```

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

```
PATH=${install-dir}/bin:${qt-install-dir}/bin:$PATH

# Act appropriately if LD_LIBRARY_PATH is not already set.
```

```
if [ -z $LD_LIBRARY_PATH ]; then
  LD_LIBRARY_PATH=${install-dir}/lib
else
  LD_LIBRARY_PATH=${install-dir}/lib:$LD_LIBRARY_PATH
fi

export 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-git"

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

> **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="${install-dir}/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 ${install-dir}/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 sddm, gdm, or xdm.

Now start your fresh KDE: in BSD and Linux® 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 fist 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 sddm, have an option to use a 'Custom Session' when you login. With this option, your 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 Git 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.