



**XRootD**

# Contributing to XRootD

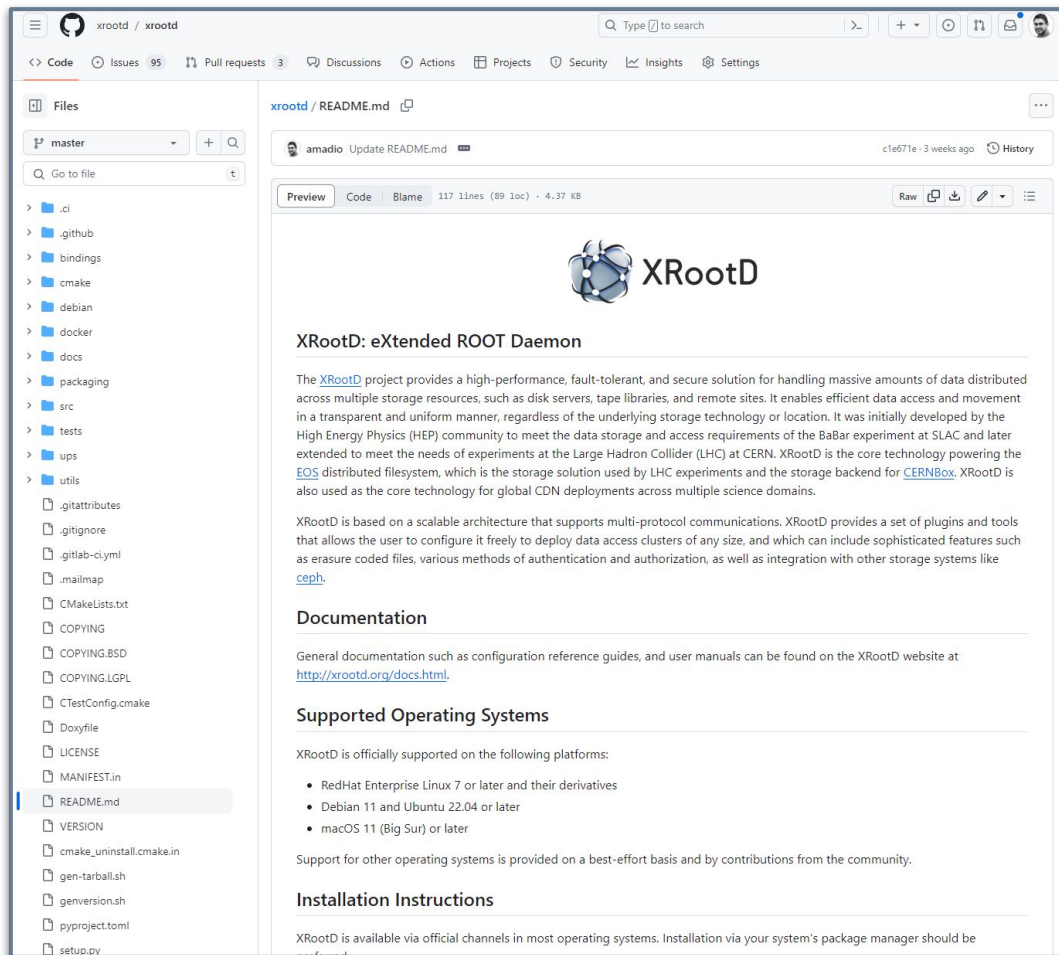
# Overview

- ▶ **XRootD repository on GitHub**
- ▶ **Development Workflow**
- ▶ **Install XRootD build dependencies**
- ▶ **Configure, build, and run tests locally**
- ▶ **Submit test results and coverage to CDash**
- ▶ **Build and test XRootD on another platform**
- ▶ **Report a bug and/or ask a question (use Discussions)**
- ▶ **Use GitHub Actions to build DEB/RPM packages**

# XRootD on GitHub

- ▶ **New README in Markdown**
- ▶ **GitHub Actions**
  - **Continuous Integration**
  - **RPM / DEB Packages**
  - **Python wheels**
  - **QEMU cross-platform**
- ▶ **CTest script**
- ▶ **CDash Dashboard**

<https://my.cdash.org/index.php?project=XRootD>



The screenshot shows the GitHub repository for XRootD. The left sidebar displays the file tree for the 'master' branch, with 'README.md' selected. The main content area shows the README file, which includes the XRootD logo, the title 'XRootD: eXtended ROOT Daemon', and a detailed description of the project. The description states that XRootD is a high-performance, fault-tolerant, and secure solution for handling massive amounts of data distributed across multiple storage resources. It also mentions that XRootD is based on a scalable architecture that supports multi-protocol communications and provides a set of plugins and tools for deployment. The README also includes sections for 'Documentation' and 'Supported Operating Systems', listing supported platforms like RedHat Enterprise Linux, Debian, Ubuntu, and macOS.

xrootd / xrootd

Code Issues 95 Pull requests 3 Discussions Actions Projects Security Insights Settings

Files

master + Q

Go to file t

- > .ci
- > .github
- > bindings
- > cmake
- > debian
- > docker
- > docs
- > packaging
- > src
- > tests
- > ups
- > utils
  - └─ .gitattributes
  - └─ .gitignore
  - └─ .gitlab-ci.yml
  - └─ .mailmap
  - └─ CMakeLists.txt
  - └─ COPYING
  - └─ COPYING.BSD
  - └─ COPYING.LGPL
  - └─ CTestConfig.cmake
  - └─ Doxyfile
  - └─ LICENSE
  - └─ MANIFEST.in
  - └─ README.md
  - └─ VERSION
  - └─ cmake\_uninstall.cmake.in
  - └─ gen-tarball.sh
  - └─ genversion.sh
  - └─ pyproject.toml
  - └─ setup.py

xrootd / README.md

amadio Update README.md c1e671e · 3 weeks ago History

Preview Code Blame 117 lines (89 loc) · 4.37 KB Raw Copy Edit

## XRootD

### XRootD: eXtended ROOT Daemon

The [XRootD](#) project provides a high-performance, fault-tolerant, and secure solution for handling massive amounts of data distributed across multiple storage resources, such as disk servers, tape libraries, and remote sites. It enables efficient data access and movement in a transparent and uniform manner, regardless of the underlying storage technology or location. It was initially developed by the High Energy Physics (HEP) community to meet the data storage and access requirements of the BaBar experiment at SLAC and later extended to meet the needs of experiments at the Large Hadron Collider (LHC) at CERN. XRootD is the core technology powering the [EOS](#) distributed filesystem, which is the storage solution used by LHC experiments and the storage backend for [CERNBox](#). XRootD is also used as the core technology for global CDN deployments across multiple science domains.

XRootD is based on a scalable architecture that supports multi-protocol communications. XRootD provides a set of plugins and tools that allows the user to configure it freely to deploy data access clusters of any size, and which can include sophisticated features such as erasure coded files, various methods of authentication and authorization, as well as integration with other storage systems like [ceph](#).

### Documentation

General documentation such as configuration reference guides, and user manuals can be found on the XRootD website at <http://xrootd.org/docs.html>.

### Supported Operating Systems

XRootD is officially supported on the following platforms:

- RedHat Enterprise Linux 7 or later and their derivatives
- Debian 11 and Ubuntu 22.04 or later
- macOS 11 (Big Sur) or later

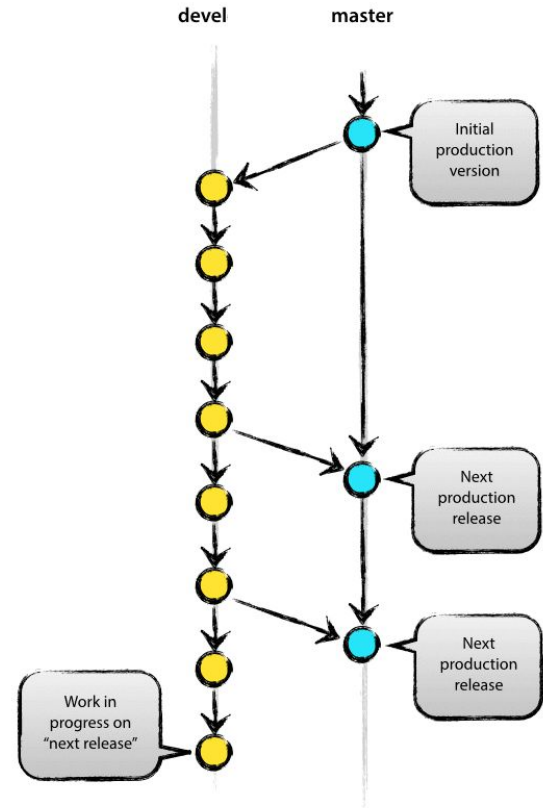
Support for other operating systems is provided on a best-effort basis and by contributions from the community.

### Installation Instructions

XRootD is available via official channels in most operating systems. Installation via your system's package manager should be

# Development Workflow and Release Management

- ▶ **Current development workflow**
  - Use **devel** branch as work in progress for next release
  - Release manager applies commits from **devel** to **master**, writes out release notes and tags releases.
- ▶ **Advantages**
  - Easy for contributors
  - Stability on **master** branch
  - Linear git history on **master** branch
  - No rebase conflicts on release notes file
- ▶ **Disadvantages**
  - No automatic closing of GitHub issues
  - Rebase on **devel** confuses GitHub pull requests



# Install Dependencies and Build XRootD Packages

## AlmaLinux 8 / 9

```
$ sudo dnf install -y epel-release # not needed on Fedora
$ sudo dnf install -y dnf-plugins-core rpmdevtools
$ sudo dnf config-manager --set-enabled powerools # Alma 8
$ sudo dnf config-manager --set-enabled crb # Alma 9
$ sudo dnf install -y git
$ git clone https://github.com/xrootd/xrootd
$ cd xrootd
$ sudo dnf builddep -y xrootd.spec
$ spectool -g -R xrootd.spec
$ rpmbuild -bb xrootd.spec
```

## Fedora Linux

```
$ sudo dnf install -y dnf-plugins-core rpmdevtools
$ sudo dnf install -y git
$ git clone https://github.com/xrootd/xrootd
$ cd xrootd
$ sudo dnf builddep -y xrootd.spec
$ spectool -g -R xrootd.spec
$ rpmbuild -bb xrootd.spec
```

## Installing:

```
$ dnf install -y ~/rpmbuild/RPMS/*//*.rpm
```

## Debian / Ubuntu

```
$ sudo apt update
$ sudo apt install -y build-essential devscripts
$ sudo apt install -y equivs # only needed on Ubuntu
$ sudo apt install -y git
$ git clone https://github.com/xrootd/xrootd
$ cd xrootd
$ export V=$(./genversion.sh --sanitize)
$ dch --create --package xrootd -v ${V} -M "XRootD ${V}"
$ mk-build-deps --install --remove -s sudo debian/control
$ debuild --no-tgz-check -- binary-arch
```

## Installing:

```
$ sudo apt install ../*.deb
```

# Building XRootD Python bindings

## Create Python Source Distribution for PyPI

```
$ git clone https://github.com/xrootd/xrootd
$ cd xrootd
$ python3 -m build --sdist
```

Alternatively,

```
$ python3 setup.py sdist
```

## Building XRootD Client as Python Package

```
$ python3 -m pip wheel .
Processing /home/amadio/src/xrootd
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: xrootd
  Building wheel for xrootd (pyproject.toml) ... done
  Created wheel for xrootd:
    filename=xrootd-5.7.1-cp312-cp312-linux_x86_64.whl
    size=69675785
  Stored in directory:
/tmp/pip-ephem-wheel-cache-mbo6n6q2/wheels/cf/67/3c/514b21ddc
8aaad94bc31ed5e1d94210de6c78816039640aa90
Successfully built xrootd
```

## Build Python Bindings Against Pre-Installed XRootD

```
$ cd bindings/python
$ python3 -m pip wheel .
```

## Build Python Bindings and Install without Wheel

```
$ python3 -m pip install --use-pep517 .
```

## Build Python Bindings with CMake

```
$ cmake -S xrootd -B build -DINSTALL_PYTHON_BINDINGS=1 ...
$ cmake --build build
$ cmake --install build
```

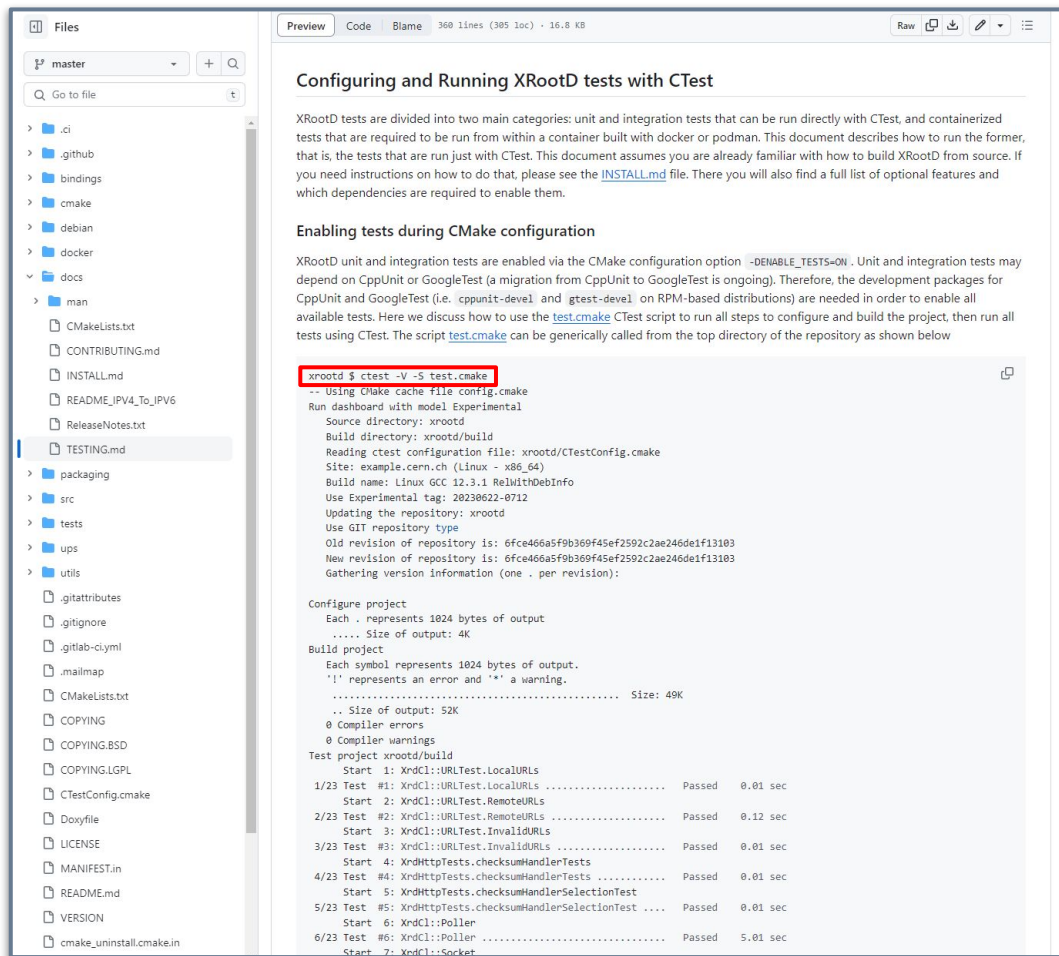
Notes:

- Cannot distribute binary wheels because of OpenSSL
- Not quite willing to statically link due to security

# XRootD on GitHub

- ▶ New README in Markdown
- ▶ GitHub Actions
  - Continuous Integration
  - RPM / DEB Packages
  - Python wheels
  - QEMU cross-platform
- ▶ CTest script
- ▶ CDash Dashboard

<https://my.cdash.org/index.php?project=XRootD>



The screenshot shows a GitHub repository interface. On the left is a file explorer for the 'master' branch, listing various files and folders such as '.ci', '.github', 'bindings', 'cmake', 'debian', 'docker', 'docs', 'man', 'CMakeLists.txt', 'CONTRIBUTING.md', 'INSTALL.md', 'README\_IPV4\_To\_IPV6', 'ReleaseNotes.txt', 'TESTING.md', 'packaging', 'src', 'tests', 'ups', 'utils', '.gitattributes', '.gitignore', '.gitlab-ci.yml', '.mailmap', 'CMakeLists.txt', 'COPYING', 'COPYING.BSD', 'COPYING.LGPL', 'CTestConfig.cmake', 'Doxyfile', 'LICENSE', 'MANIFEST.in', 'README.md', 'VERSION', and 'cmake\_uninstall.cmake.in'. The 'tests' folder is selected.

On the right is a terminal window showing the output of a CTest command. The command is `xrootd $ ctest -V -S test.cmake`. The output shows the configuration of the project, including the source directory, build directory, and the use of experimental features. It then lists the test results:

```
Test project xrootd/build
Start 1: XrdCl::URLTest.LocalURLs
1/23 Test #1: XrdCl::URLTest.LocalURLs ..... Passed   0.01 sec
Start 2: XrdCl::URLTest.RemoteURLs
2/23 Test #2: XrdCl::URLTest.RemoteURLs ..... Passed   0.12 sec
Start 3: XrdCl::URLTest.InvalidURLs
3/23 Test #3: XrdCl::URLTest.InvalidURLs ..... Passed   0.01 sec
Start 4: XrdHttpTests.checksumHandlerTests
4/23 Test #4: XrdHttpTests.checksumHandlerTests ..... Passed   0.01 sec
Start 5: XrdHttpTests.checksumHandlerSelectionTest
5/23 Test #5: XrdHttpTests.checksumHandlerSelectionTest ..... Passed   0.01 sec
Start 6: XrdCl::Poller
6/23 Test #6: XrdCl::Poller ..... Passed   5.01 sec
Start 7: XrdCl::Socket
```

# XRootD on GitHub

- ▶ New README in Markdown
- ▶ GitHub Actions
  - Continuous Integration
  - RPM / DEB Packages
  - Python wheels
  - QEMU cross-platform
- ▶ CTest script
- ▶ CDash Dashboard

<https://my.cdash.org/index.php?project=XRootD>

**Customizing the Build**

**Selecting a build type, compile flags, optional features, etc**

Since the script is targeted for usage with continuous integration, it tries to load a configuration file from the `.ci` subdirectory in the source directory. The default configuration is in the `config.cmake` file. This file is used to pre-load the CMake cache. If found, it is passed to CMake during configuration via the `-C` option. This file is a CMake script that should only contain CMake `set()` commands using the `CACHE` option to populate the cache. Some effort is made to detect and use a more specific configuration file than the generic `config.cmake` that is used by default. For example, on Ubuntu, a file named `ubuntu.cmake` will be used if present. The script also tries to detect the version of the OS and use a more specific file if found for that version. For example, on Alma Linux 8, one could use `alma1linux8.cmake` which would have higher precedence than `alma1linux.cmake`. The default `config.cmake` file will enable as many options as possible without failing if the dependencies are not installed, so it should be sufficient in most cases.

The behavior of the `test.cmake` script can also be influenced by environment variables like `CC`, `CXX`, `CXXFLAGS`, `CMAKE_ARGS`, `CMAKE_GENERATOR`, `CMAKE_BUILD_PARALLEL_LEVEL`, `CTEST_PARALLEL_LEVEL`, and `CTEST_CONFIGURATION_TYPE`. These are mostly self-explanatory and can be used to override the provided defaults. For example, to build with `clang` and use `ninja` as CMake generator, one can run:

```
xrootd $ env CC=clang CXX=clang++ CMAKE_GENERATOR=ninja ctest -V -S test.cmake
```

For performance analysis and profiling with `perf`, we recommend building with

```
xrootd $ CXXFLAGS="-fno-omit-frame-pointer" ctest -V -C RelWithDebInfo -S test.cmake
```

For enabling link-time optimizations (LTO), we recommend using

```
CXXFLAGS="-flto -Werror-odr -Werror-lto-type-mismatch -Werror-strict-aliasing"
```

This turns some important warnings into errors to avoid potential runtime issues with LTO. Please see GCC's manual page for descriptions of each of the warnings above. XRootD also support using address and thread sanitizers, via the options `-DENABLE_ASAN=ON` and `-DENABLE_TSAN=ON`, respectively. These should be enabled using `CMAKE_ARGS`, as shown below

```
$ env CMAKE_ARGS="-DENABLE_TSAN=1" ctest -V -S test.cmake
```

Note that options passed by setting `CMAKE_ARGS` in the environment have higher precedence than what is in the pre-loaded cache file, so this method can be used to override the defaults without having to edit the pre-loaded cache file.

**Enabling coverage, memory checking, and static analysis**

The `test.cmake` has several options that allow the developer to customize the build being tested. The main options are shown in the table below:

Option	Description
<code>-DCOVERAGE=ON</code>	Enables test coverage analysis with <code>gcov</code>
<code>-DMEMCHECK=ON</code>	Enables memory checking with <code>valgrind</code>



# How to use *test.cmake* to run the XRootD test suite

```
# Simplest case, configure, build, and test
$ ctest -VV -S test.cmake

# Build in Debug mode
$ ctest -VV -C Debug -S test.cmake

# Run static analysis with clang-tidy
$ ctest -VV -DSTATIC_ANALYSIS=1 -S test.cmake

# Perform memory checking with valgrind on all tests
$ ctest -VV -DMEMCHECK=1 -S test.cmake

# Build in Debug mode and create coverage report
$ ctest -VV -C Debug -DCOVERAGE=1 -S test.cmake

# Use clang compiler to build
$ env CC=clang CXX=clang++ ctest -VV -S test.cmake

# Build in Release mode and submit test results to CDash
$ ctest -VV -C Release -DCDASH=1 -S test.cmake

# Use custom configuration for CMake (build, but don't test)
$ env CMAKE_ARGS="-DENABLE_TESTS=0" ctest -VV -S test.cmake

Please see docs/TESTING.md on GitHub for more information.
```

```
~/src/xrootd $ ctest -V -S test.cmake
-- Using CMake cache file gentoo.cmake
Run dashboard with model Experimental
Source directory: /home/amadio/src/xrootd
Build directory: /home/amadio/src/xrootd/build
Group: Experimental
Reading ctest configuration file: /home/amadio/src/xrootd/CTestConfig.cmake
Site: gentoo.cern.ch
Build name: Gentoo Linux GCC 14.2.1 RelWithDebInfo
Use Experimental tag: 20240904-1451
Updating the repository: /home/amadio/src/xrootd
Use GIT repository type
Old revision of repository is: 237681febbda92020883249a2def24e88a664b28
New revision of repository is: 237681febbda92020883249a2def24e88a664b28
Gathering version information (one . per revision):

Configure project
Each . represents 1024 bytes of output
..... Size of output: 4K
Build project
Each symbol represents 1024 bytes of output.
..... Size: 49K
..... Size of output: 57K
0 Compiler errors
0 Compiler warnings
Test project /home/amadio/src/xrootd/build
Start 115: XRootD::start
Start 1: XrdCl::URLTest_LocalURLs
Start 2: XrdCl::URLTest_RemoteURLs
Start 3: XrdCl::URLTest_InvalidURLs
Start 4: XrdCl::PollerTest_FunctionTest
Start 5: XrdCl::SocketTest_TransferTest
Start 6: XrdCl::UtilsTest_AnyTest
Start 7: XrdCl::UtilsTest_TaskManagerTest
Start 8: XrdCl::UtilsTest_SIDManagerTest
Start 9: XrdCl::UtilsTest_PropertyListTest
Start 75: XrdHttpTests_checksumHandlerTests
1/120 Test #115: XRootD::start ..... Passed 0.02 sec
2/120 Test #1: XrdCl::URLTest_LocalURLs ..... Passed 0.02 sec
3/120 Test #3: XrdCl::URLTest_InvalidURLs ..... Passed 0.02 sec
... (many more lines)
119/120 Test #90: XrdCl::UtilsTest ..... Passed 8.02 sec
Start 120: XRootD::cluster::stop
120/120 Test #120: XRootD::cluster::stop ..... Passed 0.04 sec

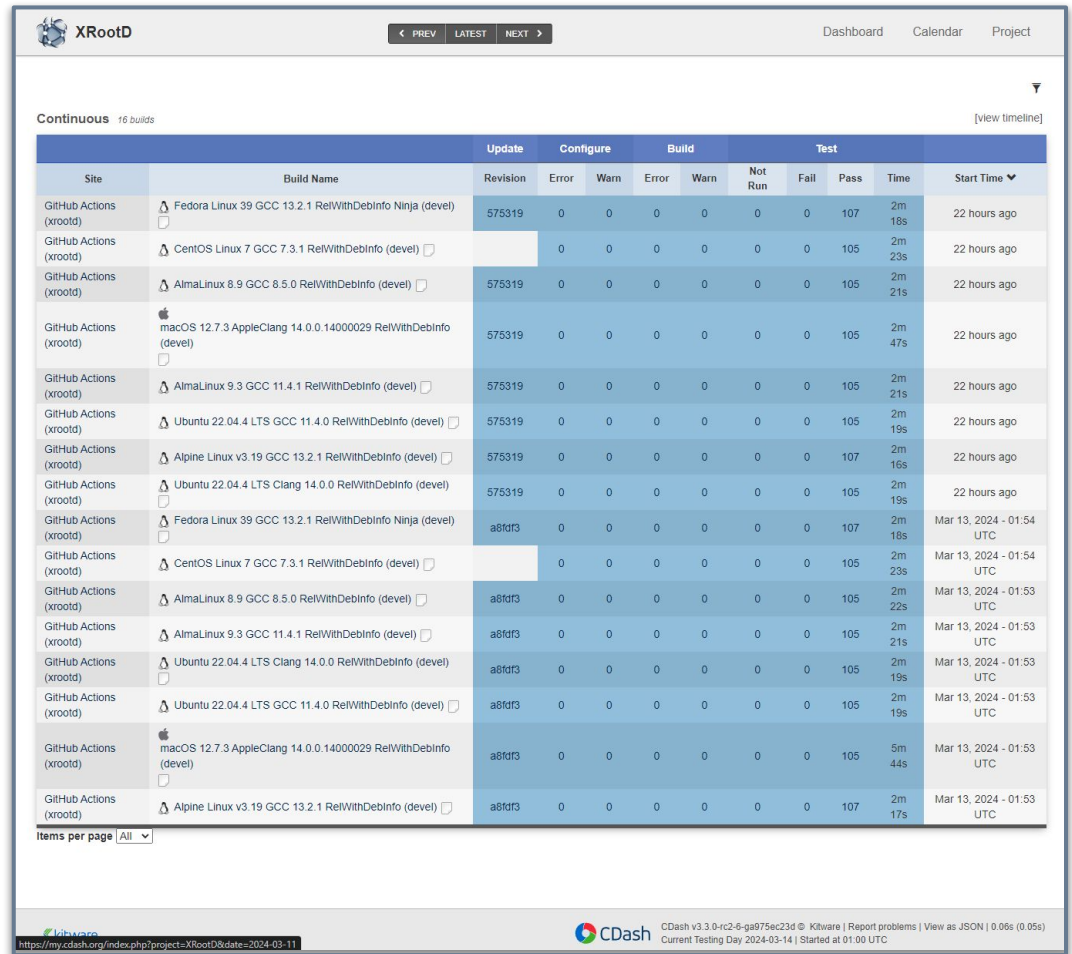
100% tests passed, 0 tests failed out of 120

Total Test time (real) = 143.62 sec
```

# XRootD on GitHub

- ▶ New README in Markdown
- ▶ GitHub Actions
  - Continuous Integration
  - RPM / DEB Packages
  - Python wheels
  - QEMU cross-platform
- ▶ CTest script
- ▶ CDash Dashboard

<https://my.cdash.org/index.php?project=XRootD>



The screenshot shows the XRootD CDash dashboard. At the top, there are navigation links for 'PREV', 'LATEST', and 'NEXT'. The main content area is titled 'Continuous' and shows a table of 16 builds. The table has columns for 'Site', 'Build Name', 'Update', 'Configure', 'Build', 'Test', and 'Start Time'. The 'Test' column is further divided into 'Not Run', 'Fail', 'Pass', and 'Time'. The builds are listed in descending order of start time, with the most recent build starting on Mar 13, 2024 at 01:54 UTC. The table shows that all builds passed successfully.

Site	Build Name	Update	Configure		Build		Test			Start Time	
		Revision	Error	Warn	Error	Warn	Not Run	Fail	Pass		Time
GitHub Actions (xrootd)	Fedora Linux 39 GCC 13.2.1 RelWithDebInfo Ninja (devel)	575319	0	0	0	0	0	0	107	2m 18s	22 hours ago
GitHub Actions (xrootd)	CentOS Linux 7 GCC 7.3.1 RelWithDebInfo (devel)		0	0	0	0	0	0	105	2m 23s	22 hours ago
GitHub Actions (xrootd)	AlmaLinux 8.9 GCC 8.5.0 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	105	2m 21s	22 hours ago
GitHub Actions (xrootd)	macOS 12.7.3 AppleClang 14.0.0.14000029 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	105	2m 47s	22 hours ago
GitHub Actions (xrootd)	AlmaLinux 9.3 GCC 11.4.1 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	105	2m 21s	22 hours ago
GitHub Actions (xrootd)	Ubuntu 22.04.4 LTS GCC 11.4.0 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	105	2m 19s	22 hours ago
GitHub Actions (xrootd)	Alpine Linux v3.19 GCC 13.2.1 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	107	2m 16s	22 hours ago
GitHub Actions (xrootd)	Ubuntu 22.04.4 LTS Clang 14.0.0 RelWithDebInfo (devel)	575319	0	0	0	0	0	0	105	2m 19s	22 hours ago
GitHub Actions (xrootd)	Fedora Linux 39 GCC 13.2.1 RelWithDebInfo Ninja (devel)	a8fd3	0	0	0	0	0	0	107	2m 18s	Mar 13, 2024 - 01:54 UTC
GitHub Actions (xrootd)	CentOS Linux 7 GCC 7.3.1 RelWithDebInfo (devel)		0	0	0	0	0	0	105	2m 23s	Mar 13, 2024 - 01:54 UTC
GitHub Actions (xrootd)	AlmaLinux 8.9 GCC 8.5.0 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	105	2m 22s	Mar 13, 2024 - 01:53 UTC
GitHub Actions (xrootd)	AlmaLinux 9.3 GCC 11.4.1 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	105	2m 21s	Mar 13, 2024 - 01:53 UTC
GitHub Actions (xrootd)	Ubuntu 22.04.4 LTS Clang 14.0.0 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	105	2m 19s	Mar 13, 2024 - 01:53 UTC
GitHub Actions (xrootd)	Ubuntu 22.04.4 LTS GCC 11.4.0 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	105	2m 19s	Mar 13, 2024 - 01:53 UTC
GitHub Actions (xrootd)	macOS 12.7.3 AppleClang 14.0.0.14000029 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	105	5m 44s	Mar 13, 2024 - 01:53 UTC
GitHub Actions (xrootd)	Alpine Linux v3.19 GCC 13.2.1 RelWithDebInfo (devel)	a8fd3	0	0	0	0	0	0	107	2m 17s	Mar 13, 2024 - 01:53 UTC

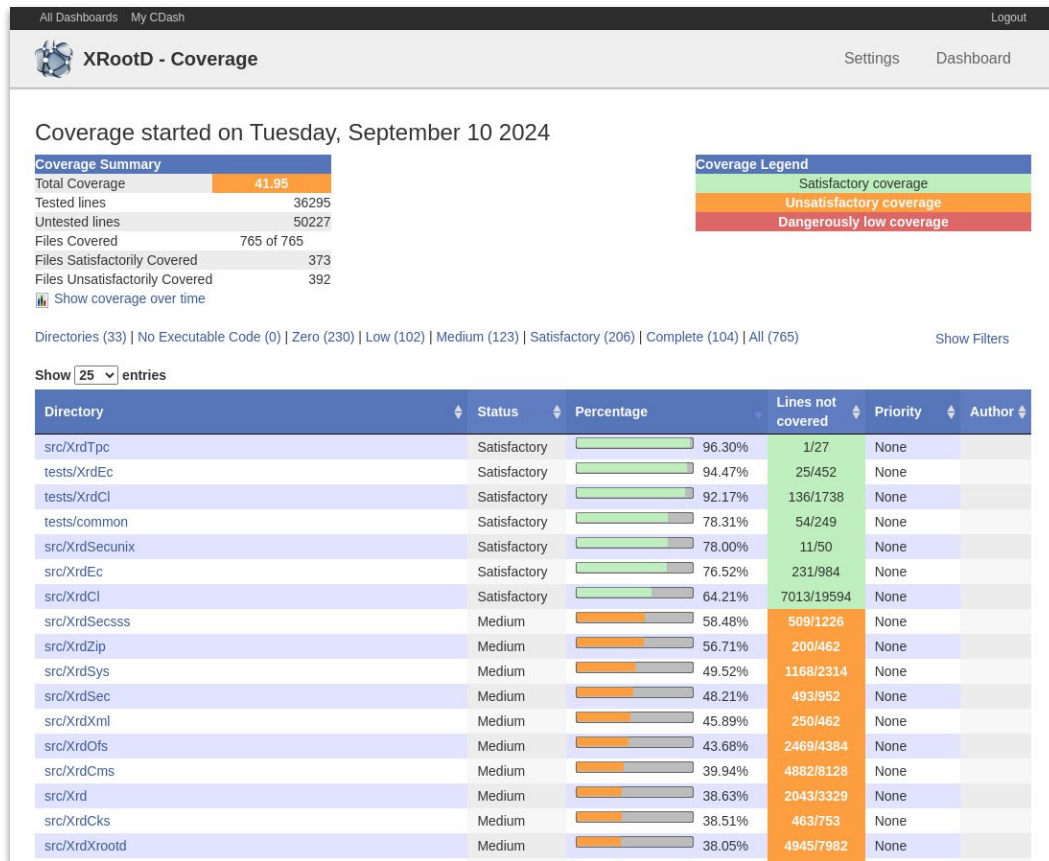
Items per page: All

CDash v3.3.0-rc2-6-ga975ec23d © Kitware | Report problems | View as JSON | 0.06s (0.05s)  
Current Testing Day 2024-03-14 | Started at 01:00 UTC

# XRootD on GitHub

- ▶ New README in Markdown
- ▶ GitHub Actions
  - Continuous Integration
  - RPM / DEB Packages
  - Python wheels
  - QEMU cross-platform
- ▶ CTest script
- ▶ CDash Dashboard

<https://my.cdash.org/index.php?project=XRootD>



# XRootD on GitHub

- ▶ New README in Markdown
- ▶ GitHub Actions
  - Continuous Integration
  - RPM / DEB Packages
  - Python wheels
  - QEMU cross-platform
- ▶ CTest script
- ▶ CDash Dashboard

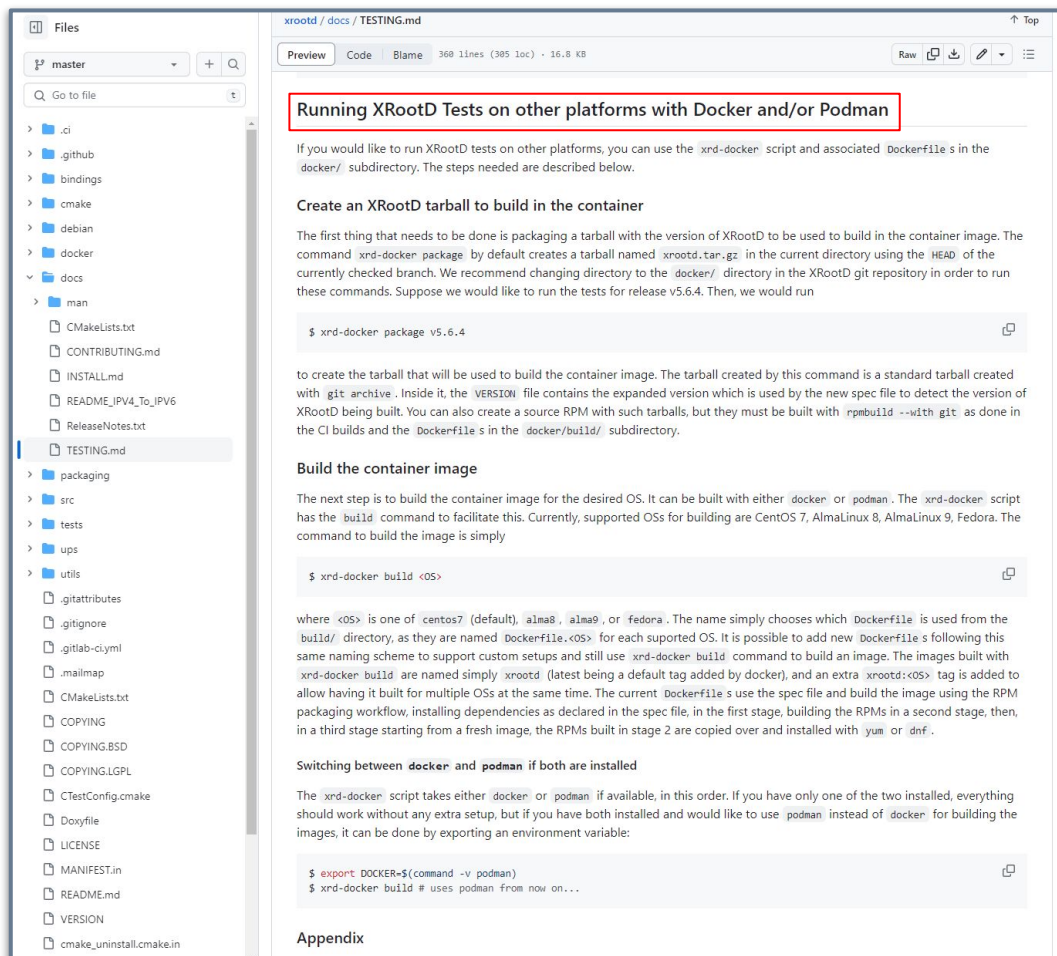
<https://my.cdash.org/index.php?project=XRootD>

```
106 | //-----  
107 | // Examine an incoming message, and decide on the action to be taken  
108 | //-----  
109 2560 | uint16_t XRootDMsgHandler::Examine( std::shared_ptr<Message> &msg )  
110 | {  
111 |     //-----  
112 |     // if the MsgHandler is already being used to process another request  
113 |     // (kXR_oksofar) we need to wait  
114 |     //-----  
115 2560 |     if( pOksofarAsAnswer )  
116 |     {  
117 120 |         XrdSysCondVarHelper lck( pCV );  
118 232 |         while( pResponse ) pCV.Wait();  
119 120 |     }  
120 |     else  
121 |     {  
122 2440 |         if( pResponse )  
123 |         {  
124 0 |             Log *log = DefaultEnv::GetLog();  
125 0 |             log->Warning( ExDbgMsg, "[%s] MsgHandler is examining a response although "  
126 |                 "it already owns a response: 0x%x (message: %s ).",  
127 0 |                 pUrl.GetHostId().c_str(), this,  
128 0 |                 pRequest->GetObfuscatedDescription().c_str() );  
129 |         }  
130 |     }  
131 |  
132 2560 |     if( msg->GetSize() < 8 )  
133 0 |         return Ignore;  
134 |  
135 2560 |     ServerResponse *rsp = (ServerResponse *)msg->GetBuffer();  
136 2560 |     ClientRequest *req = (ClientRequest *)pRequest->GetBuffer();  
137 2560 |     uint16_t status = 0;  
138 2560 |     uint32_t dlen = 0;  
139 |  
140 |     //-----  
141 |     // We only care about async responses, but those are extracted now  
142 |     // in the SocketHandler.  
143 |     //-----  
144 2560 |     if( rsp->hdr.status == kXR_attn )  
145 |     {  
146 0 |         return Ignore;  
147 |     }  
148 |     //-----  
149 |     // We got a sync message - check if it belongs to us  
150 |     //-----  
151 |     else  
152 |     {  
153 2560 |         if( rsp->hdr.streamid[0] != req->header.streamid[0] ||
```

# XRootD on GitHub

- ▶ New README in Markdown
- ▶ GitHub Actions
  - Continuous Integration
  - RPM / DEB Packages
  - Python wheels
  - QEMU cross-platform
- ▶ CTest script
- ▶ CDash Dashboard

<https://my.cdash.org/index.php?project=XRootD>



The screenshot shows the GitHub web interface for the XRootD repository, specifically the `docs/TESTING.md` file. The left sidebar shows the repository's file structure, with `docs/TESTING.md` selected. The main content area displays the file's metadata (360 lines, 305 loc, 16.8 KB) and a red-bordered title: "Running XRootD Tests on other platforms with Docker and/or Podman".

**Running XRootD Tests on other platforms with Docker and/or Podman**

If you would like to run XRootD tests on other platforms, you can use the `xrd-docker` script and associated `Dockerfile`s in the `docker/` subdirectory. The steps needed are described below.

### Create an XRootD tarball to build in the container

The first thing that needs to be done is packaging a tarball with the version of XRootD to be used to build in the container image. The command `xrd-docker package` by default creates a tarball named `xrootd.tar.gz` in the current directory using the `HEAD` of the currently checked branch. We recommend changing directory to the `docker/` directory in the XRootD git repository in order to run these commands. Suppose we would like to run the tests for release v5.6.4. Then, we would run

```
$ xrd-docker package v5.6.4
```

to create the tarball that will be used to build the container image. The tarball created by this command is a standard tarball created with `git archive`. Inside it, the `VERSION` file contains the expanded version which is used by the new spec file to detect the version of XRootD being built. You can also create a source RPM with such tarballs, but they must be built with `rpmbuild --with git` as done in the CI builds and the `Dockerfile`s in the `docker/build/` subdirectory.

### Build the container image

The next step is to build the container image for the desired OS. It can be built with either `docker` or `podman`. The `xrd-docker` script has the `build` command to facilitate this. Currently, supported OSs for building are CentOS 7, AlmaLinux 8, AlmaLinux 9, Fedora. The command to build the image is simply

```
$ xrd-docker build <OS>
```

where `<OS>` is one of `centos7` (default), `alma8`, `alma9`, or `fedora`. The name simply chooses which `Dockerfile` is used from the `build/` directory, as they are named `Dockerfile.<OS>` for each supported OS. It is possible to add new `Dockerfile`s following this same naming scheme to support custom setups and still use `xrd-docker build` command to build an image. The images built with `xrd-docker build` are named simply `xrootd` (latest being a default tag added by `docker`), and an extra `xrootd:<OS>` tag is added to allow having it built for multiple OSs at the same time. The current `Dockerfile`s use the spec file and build the image using the RPM packaging workflow, installing dependencies as declared in the spec file, in the first stage, building the RPMs in a second stage, then, in a third stage starting from a fresh image, the RPMs being in stage 2 are copied over and installed with `yum` or `dnf`.

### Switching between `docker` and `podman` if both are installed

The `xrd-docker` script takes either `docker` or `podman` if available, in this order. If you have only one of the two installed, everything should work without any extra setup, but if you have both installed and would like to use `podman` instead of `docker` for building the images, it can be done by exporting an environment variable:

```
$ export DOCKER=$(command -v podman)
$ xrd-docker build # uses podman from now on...
```

### Appendix

GitHub Actions interface for repository `xrootd / xrootd`. The `Actions` tab is active, showing workflow runs for `QEMU`.

**Navigation:** Code, Issues (95), Pull requests (3), Discussions, **Actions**, Projects, Security, Insights, Settings.

**Workflow Runs:** 1 workflow run. Filter: Filter workflow runs. Run workflow button.

**Workflow Details:** QEMU (QEMU.yml). This workflow has a `workflow_dispatch` event trigger.

**Run Configuration:**

- Use workflow from: Branch: master
- OS: fedora
- Architecture: s390x
- Run workflow button

**Left Sidebar (Actions):** All workflows, Workflows, CI, DEB, Python, **QEMU**, RPM, Management, Caches, Runners.

Files

master

Go to file

- > .ci
- ▼ .github/workflows
  - Cl.yml
  - DEB.yml
  - QEMU.yml
  - RPM.yml
  - python.yml
- > bindings
- > cmake
- > debian
- > docker
- > docs
- > packaging
- > src
- > tests
- > ups
- > utils

xrootd / .github / workflows / QEMU.yml

Code Blame 57 lines (49 loc) · 1.16 KB

Raw Copy Download Edit

```
30
31 concurrency:
32   group: ${{ github.workflow }}-${{ github.ref }}-${{ inputs.os }}-${{ inputs.arch }}
33   cancel-in-progress: true
34
35 defaults:
36   run:
37     shell: bash
38
39 env:
40   DOCKER: podman
41
42 jobs:
43   buildx:
44     name: QEMU (${{ inputs.os }}-${{ inputs.arch }})
45     runs-on: ubuntu-latest
46
47     steps:
48     - name: Clone repository
49       uses: actions/checkout@v3
50       with:
51         fetch-depth: 0
52
53     - name: Setup QEMU for cross-building images
54       run: docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
55
56     - name: Cross-build container with docker/podman buildx
57       run: cd docker && ./xrd-docker buildx ${{ inputs.os }} ${{ inputs.arch }}
```

```
docker $ xrd-docker
xrd-docker [COMMAND] [ARGS]
```

COMMANDS:

```
clean          -- remove tarball created by package command
package [VERSION] -- create xrootd.tar.gz tarball (VERSION=HEAD by default)
build [OS]      -- build docker image based on OS: centos7 (default), alma8, alma9
buildx [OS] [ARCH] -- cross-build docker image based on OS/ARCH pair. Supported architectures
                  are amd64, aarch64, ppc64le, s390x (big-endian). Default OS is fedora.
                  You can see supported platforms with docker buildx inspect --bootstrap.
qemu           -- setup QEMU to be able to run cross-builds with buildx command.
```

Note: The test suite runs automatically during the container builds

```
docker $ xrd-docker package
Creating tarball for XRootD v5.7.1
```

```
docker $ ls
```

```
build xrd-docker xrootd.tar.gz
```

```
docker $ ls build
```

```
Dockerfile.alma8 Dockerfile.alma9 Dockerfile.centos7 Dockerfile.debian Dockerfile.fedora Dockerfile.ubuntu
```

```
docker $ xrd-docker qemu
```

```
Setting /usr/bin/qemu-alpha-static as binfmt interpreter for alpha
```

```
Setting /usr/bin/qemu-arm-static as binfmt interpreter for arm
```

```
Setting /usr/bin/qemu-sparc-static as binfmt interpreter for sparc
```

```
Setting /usr/bin/qemu-sparc32plus-static as binfmt interpreter for sparc32plus
```

```
Setting /usr/bin/qemu-sparc64-static as binfmt interpreter for sparc64
```

```
Setting /usr/bin/qemu-ppc-static as binfmt interpreter for ppc
```

```
Setting /usr/bin/qemu-ppc64-static as binfmt interpreter for ppc64
```

```
Setting /usr/bin/qemu-ppc64le-static as binfmt interpreter for ppc64le
```

```
Setting /usr/bin/qemu-m68k-static as binfmt interpreter for m68k
```

```
Setting /usr/bin/qemu-s390x-static as binfmt interpreter for s390x
```

```
Setting /usr/bin/qemu-aarch64-static as binfmt interpreter for aarch64
```

```
Setting /usr/bin/qemu-aarch64_be-static as binfmt interpreter for aarch64_be
```

```
...
```



```

docker $ xrd-docker buildx alma9 s390x
[+] Building 1634.2s (16/16) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile.alma9           0.0s
=> => transferring dockerfile: 890B                                 0.0s
=> [internal] load metadata for docker.io/library/almalinux:9       1.2s
=> [auth] library/almalinux:pull token for registry-1.docker.io    0.0s
=> [internal] load .dockerignore                                    0.0s
=> => transferring context: 46B                                     0.0s
=> [ 1/10] FROM docker.io/library/almalinux:9@sha256:ff4f72c2c65badbc7deea85a035d13b6fc5160b97777939e97479921c57a3cd7 4.2s
=> => resolve docker.io/library/almalinux:9@sha256:ff4f72c2c65badbc7deea85a035d13b6fc5160b97777939e97479921c57a3cd7 0.0s
=> => sha256:ff4f72c2c65badbc7deea85a035d13b6fc5160b97777939e97479921c57a3cd7 4.70kB / 4.70kB 0.0s
=> => sha256:3a1bee15a9da3a05268ecb75b750749c6391a1a04b28877e2751ed8837b242df 1.03kB / 1.03kB 0.0s
=> => sha256:24bf7a4a5d87e25d06f490b4d48c13a1945e3178a6558844edd9b0e09876b2e3 598B / 598B 0.0s
=> => sha256:6a98952f4df22ffcc7ebe6935496f5d568398f92da9469584491852b68f03968 66.58MB / 66.58MB 1.9s
=> => extracting sha256:6a98952f4df22ffcc7ebe6935496f5d568398f92da9469584491852b68f03968 2.2s
=> [internal] load build context                                    0.0s
=> => transferring context: 6.86MB                                  0.0s
=> [2/10] RUN dnf install -y dnf-plugins-core epel-release rpmdevtools sudo && dnf config-manager --set-enabled crb 67.9s
=> [3/10] RUN groupadd xrootd && useradd -g xrootd -m xrootd        1.1s
=> [4/10] WORKDIR /home/xrootd                                     0.3s
=> [5/10] RUN rpmdev-setuptree                                     1.6s
=> [6/10] COPY xrootd.tar.gz rpmbuild/SOURCES                     0.2s
=> [7/10] RUN tar xzf rpmbuild/SOURCES/xrootd.tar.gz --strip-components=1 xrootd/xrootd.spec 1.1s
=> [8/10] RUN dnf builddep -y xrootd.spec                         205.5s
=> [9/10] RUN sudo -u xrootd rpmbuild -bb --with git xrootd.spec 1301.8s
=> [10/10] RUN yum install -y rpmbuild/RPMS/*/*.rpm              45.9s
=> exporting to image                                             3.2s
=> => exporting layers                                           3.2s
=> => writing image sha256:e36db8e9381955466db3e95d2e312fdd45bae5c4e787643f7fb20c68a1393957 0.0s
=> => naming to docker.io/library/xrootd:alma9-s390x            0.0s

```

```
docker $ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
xrootd	alma9-s390x	e36db8e93819	1 minute ago	2.61GB
multiarch/qemu-user-static	latest	3539aaa87393	19 months ago	305MB

# Issues, Discussions, Security Vulnerabilities

The screenshot shows the GitHub interface for the repository `xrootd / xrootd`. The navigation bar at the top includes links for Code, Issues (89), Pull requests (4), Discussions (highlighted with a red box), Actions, Projects, Security (194, highlighted with a red box), Insights, and Settings. A search bar is present with the placeholder text "Type to search".

The main content area features a welcome message: "Welcome to XRootD Discussions!" with a sub-header "Announcements · amadio". Below this is a search bar containing "is:open", a "Sort by: Latest activity" dropdown, a "Label" dropdown, a "Filter: Open" dropdown, and a "New discussion" button.

On the left, the "Categories" section lists: "View all discussions" (selected), "Announcements", "General", "Ideas", "Polls", and "Q&A".

The "Discussions" section displays a list of discussion items:

- How to implement TAPE REST API call**: A Discussion and Documentation Issue. Announced by maksiks on Apr 24 in Announcements. 15 replies.
- need help with configuration for davix-cp**: Asked by maksiks on Mar 19 in Q&A · Unanswered. 2 replies.
- Welcome to XRootD Discussions!**: Announced by amadio on Feb 16 in Announcements. 0 replies.

# GitHub Actions Overview

The screenshot shows the GitHub Actions interface for the repository `xrootd / xrootd`. The **Actions** tab is highlighted with a red box. In the left sidebar, the **Actions** section is also highlighted with a red box, and the **COV** workflow is selected and highlighted in grey. The main content area shows the **COV** workflow details, including a search bar for workflow runs, a "Help us improve GitHub Actions" banner, and a section for **51 workflow runs**. A dropdown menu for "Run workflow" is open, showing the selected branch as `Branch: master` and a **Run workflow** button. The workflow runs list includes:

- Change type in XrdSutCacheArg\_t to long...** (COV #51: Commit `ca9c703` pushed by amadio)
- Add error string to e2sMap if EBADE is ou...** (COV #50: Commit `f633e0f` pushed by abh3)
- Protect against array index out of bounds**

# GitHub Actions Overview

**Summary**

Jobs

- CentOS 7
- Alma Linux 8
- Alma Linux 9
- Fedora 40













Run details

- Usage
- Workflow file

**CentOS 7**  
Node.js 16 actions are deprecated. Please update the following actions to use Node.js 20: actio...  
[Show more](#)

**Deprecation notice: v1, v2, and v3 of the artifact actions**  
The following artifacts were uploaded using a version of actions/upload-artifact that is schedu...  
[Show more](#)

**Artifacts**  
Produced during runtime

Name	Size		
 <b>alma8</b>	44.7 MB		
 <b>alma9</b>	32.9 MB		
 <b>centos7</b>	32.2 MB		
 <b>fc39</b>	31.9 MB		

# Summary

- ▶ **Configure, build, test cycle automated with CMake (test.cmake)**
  - Low entry barrier, easy to run same thing as the CI locally on your machine
- ▶ **Continuous Integration with GitHub Actions**
  - Leverages test.cmake script and DEB/RPM packaging to keep CI description simple
- ▶ **Builds and runs tests on all supported platforms**
  - Alpine (MUSL), CentOS 7, Alma 8, Alma 9, Fedora, Ubuntu (GCC & Clang), macOS
- ▶ **CDash dashboard results to make digging through errors more easily**
- ▶ **Coverage reports in CDash as well as CodeCov.io (not covered)**
  - Make it clear which areas of the code are *not* covered by tests
- ▶ **Use static analysis to spot problems in parts of the code which are not covered by tests**
  - Clang-tidy, CodeQL, valgrind, asan, tsan, etc.



**XRootD**