Modules Compiling

EasyBuild Modules Compiling automates the process of installing scientific software modules on computing clusters or systems. It simplifies complex software builds and installations, providing a unified and automated approach.

How to Run EasyBuild Module Compiling

1. **Select Software**: Choose the software package you want to install. For example, let's say you want to install the software package "example-software".
2. **Build the Module**: Use the `eb` command followed by the name of the software package to build it. You can optionally include the `--ignore-checksums` option to ignore checksum verification. For example:

   ```
   eb -r . (--ignore-checksums) path/example-software.eb
   ```

the dot is the starting searching path for the Dependency modules. The -r stands for Robot it searches for the Dependencies starting searching path. 3. **Load the Module**: After the module is built, load it into your environment using the `module load` command. For example:

```
module load example-module.eb
```

4. **Verify Installation**: You can verify that the software is installed correctly by checking its version or running a basic command associated with it.

Example:

Suppose you want to install the software package "GROMACS" while ignoring checksum verification. Here's how you would do it:

1. **Build the Module**:

   ```
   eb -r . --ignore-checksums path/GROMACS.eb
   ```

2. **Load the Module**:

   ```
   module load GROMACS
   ```

EasyBuild Recipe Example

When writing a recipe for EasyBuild, you'll create a `.eb` file for each software package you want to build. This file contains instructions for EasyBuild on how to download, configure, build, and install the software package. Here's a basic example of what an EasyBuild recipe file might look like:

```
easyblock = 'AutotoolsMake'
name = 'example-software'
version = '1.0'
homepage = 'https://example.com'
sources = [SOURCE_TAR_GZ]

dependencies = [('GCC', '9.3.0')]

moduleclass = 'tools'

sanity_check_paths = {
    'files': [],
    'dirs': []
}

buildopts = {'toolchain': {'name': 'GCC', 'version': '9.3.0'}}

patches = [
    ('patch1.patch', 1),
    ('patch2.patch', 1)
]

moduleclass = 'tools'

preconfigopts = [
    './configure --prefix=$EBROOTEXAMPLE_SOFTWARE',
]

moduleclass = 'tools'

postinstallcmds = [
    'echo "Example software installation complete."'
]
```

The easyblock variable specifies the EasyBuild easyblock class for building the software package. Easyblocks are Python classes that handle the build process. For example:

```
easyblock = 'AutotoolsMake'
```

This line sets the easyblock class to AutotoolsMake, suitable for packages using the Autotools build system. Other common classes include CMake, PythonPackage, and MakeFile, each for different types of software packages. This choice ensures EasyBuild applies the correct build logic and commands for successful compilation and installation.

This is just a basic example, and the contents of the `.eb` file can vary depending on the specific requirements of the software package you're building.

References

- EasyBuild Documentation: https://easybuild.readthedocs.io
- EasyBuild GitHub Repository: https://github.com/easybuilders/easybuild

See Also

- High-Performance Computing (HPC)
- Environment Modules
- Lmod

External Links

- EasyBuild Website
- EasyBuild Community Wiki

Categories

- High-Performance Computing
- Software Development
- Computational Science
- Build Automation

Revision #2
Created 6 June 2024 07:49:34 by Admin
Updated 12 June 2024 13:56:31 by Admin