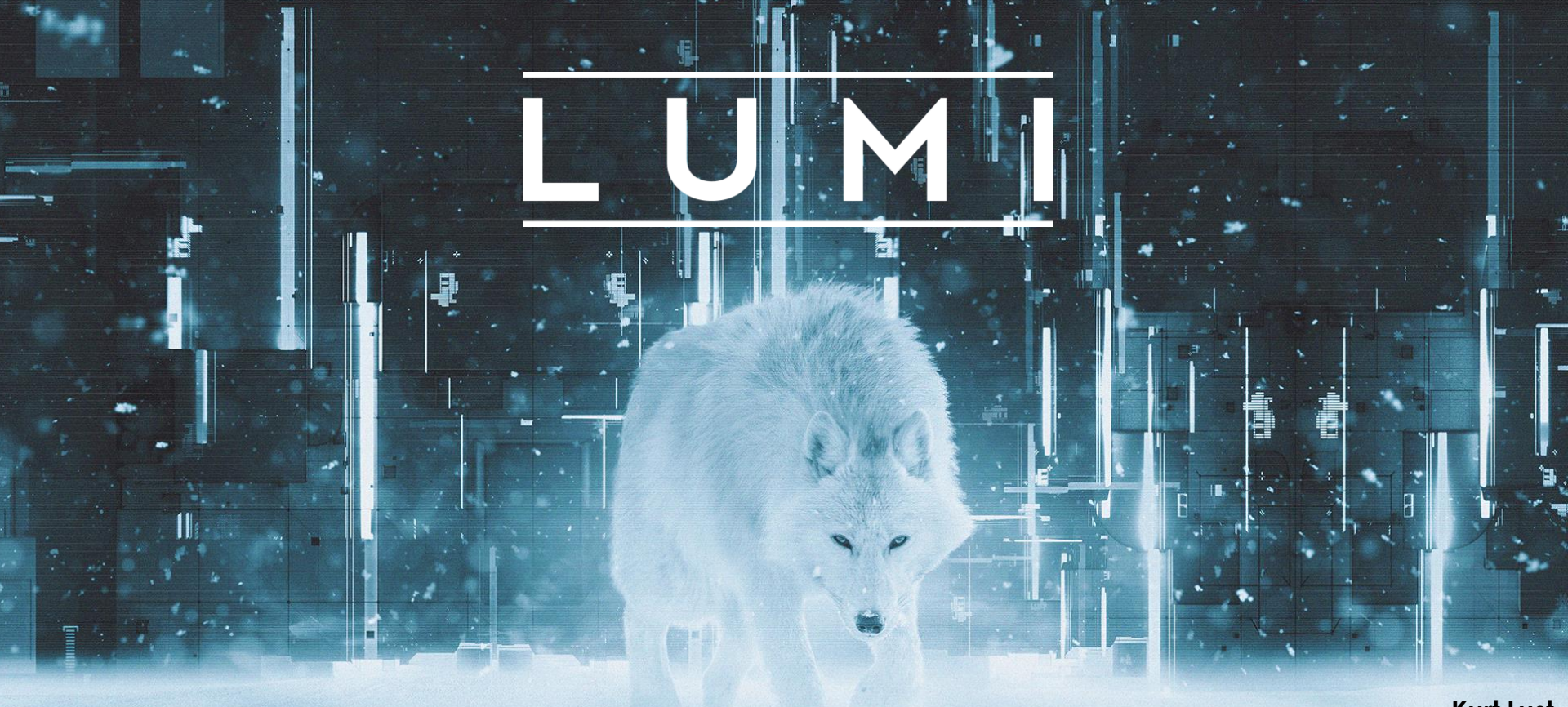


LUMI Software Stacks

Kurt Lust
LUMI User Support Team (LUST)
VSC Tier-0 support, University of Antwerp

October 2025

Software stack design considerations

LUMI

- Very leading edge and inhomogeneous machine (new interconnect, new GPU architecture with a still maturing software ecosystem, NVIDIA GPUs for visualisation, a mix of zen2 and zen3)
  - Need to remain agile
- Users that come to LUMI from 12 different channels (not counting subchannels), with different expectations
- Small central support team considering the expected number of projects and users and the tasks the support team has
  - But contributions from local support teams
- Cray Programming Environment is a key part of our system
- Users really want more and more a customised environment
  - Everybody wants a central stack as long as their software is in there but not much more
  - Look at the success of conda, Python virtual environments, containers, …

The LUMI solution

LUMI

- Software organised in extensible software stacks based on a particular release of the PE
  - Many base libraries and some packages already pre-installed
  - Easy way to install additional packages in project space
- Modules managed by Lmod
  - More powerful than the (old) Modules Environment
  - Powerful features to search for modules
- EasyBuild is our primary tool for software installations
  - But uses HPE Cray specific toolchains
  - Offer a library of installation recipes
  - User installations integrate seamlessly with the central stack
  - We do have a Spack setup but don't do development in Spack ourselves

Policies

LUMI

- Bring-your-own-license except for a selection of tools that are useful to a larger community
  - One downside of the distributed user management is that we do not even have the information needed to determine if a particular userid can use a particular software license
  - Even for software on the system, users remain responsible for checking the license!
- LUST tries to help with installations of recent software, but porting or bug fixing is not our work
  - Not all Linux or even supercomputer software will work on LUMI
  - We're too small a team to do all software installations, so don't count on us to do all the work
- Conda, (large) Python installations need to go in containers
  - Tools: lumi-container-wrapper , cotainr and SingularityCE unprivileged proot build

Organisation: Software stacks

LUMI

- **CrayEnv:** Cray environment with some additional tools pushed in through EasyBuild
- **LUMI** stacks, each one corresponding to a particular release of the PE
  - Work with the Cray PE modules, but accessed through a replacement for the PrgEnv-* modules
  - Tuned versions for the 4 types of hardware: zen2 (login, large memory nodes), zen3 (LUMI-C compute nodes), zen2 + NVIDIA GPU (visualisation partition), zen3 + MI250X (LUMI-G GPU partition)
- **spack:** Install software with Spack using compilers from the PE
  - Offered as-is for users who know Spack, but we do not do development in Spack
- Some local organisations also provide software pre-installed on LUMI
  - Look for **`Local-*` modules**
- **EESSI** may be coming next year if they can get it to work on LUMI as part of the EuroHPC federation platform, and initially likely CPU-only

Accessing the Cray PE on LUMI
3 different ways

LUMI

- Very bare environment available directly after login
  - What you can expect on a typical Cray system
  - Few tools as only the base OS image is available
  - User fully responsible for managing the target modules
- **CrayEnv**
  - "Enriched" Cray PE environment
  - Takes care of managing the target modules: (re)loading CrayEnv will reload an optimal set for the node you're on
  - Some additional tools, e.g., newer build tools (offered here and not in the bare environment as we need to avoid conflicts with other software stacks)
  - Otherwise used in the way discussed in this course

Accessing the Cray PE on LUMI
3 different ways

LUMI

- **LUMI** software stack
  - Each stack based on a particular release of the HPE Cray PE
    - Other modules are accessible but hidden from the default view
  - Better not to use the PrgEnv modules but the EasyBuild LUMI toolchains

HPE Cray PE	LUMI toolchain	
PrgEnv-cray	cpeCray	Cray Compiling Environment
PrgEnv-gnu	cpeGNU	GNU C/C++ and Fortran
PrgEnv-aocc	cpeAOCC	AMD CPU compilers (not on LUMI-G)
PrgEnv-amd	cpeAMD	AMD ROCm GPU compilers (LUMI-G only)

  - Environment in which we install most software (mostly with EasyBuild)

Accessing the Cray PE on LUMI
The LUMI software stack

LUMI

- The LUMI software stack uses two levels of modules
  - LUMI/24.03, LUMI/23.12, LUMI/23.09, LUMI/23.03, LUMI/22.08 (and more in the ccpe containers): Versions of the LUMI stack
  - partition/L, partition/C, partition/G, partition/D: To select software optimised for the respective LUMI partition
    - partition/L is for both the login nodes and the large memory nodes (4TB)
  - Hidden partition/common for software that is available everywhere, but be careful using it for your own installs
  - When (re)loaded, the LUMI module will load the best matching partition module.
  - So be careful in job scripts: When your job starts, the environment will be that of the login nodes, but if you reload the LUMI module it will be that of the compute node!

Installing software on HPC systems

LUMI

- Software on an HPC system is rarely installed from RPM
  - Generic RPMs often not optimised for the specific CPU
  - Generic RPMs may not work with the specific LUMI environment (Slingshot interconnect, kernel modules, resource manager)
  - Multi-user system so usually no "one version fits all"
  - Need a small system image as nodes are diskless
- Spack and EasyBuild are the two most popular HPC-specific software build and installation frameworks
  - Usually install from sources to adapt the software to the underlying hardware and OS
  - Installation instructions in a way that can be communicated and executed easily
  - Make software available via modules
  - Dependency handling compatible with modules

Extending the LUMI stack with EasyBuild

LUMI

- Fully integrated in the LUMI software stack
  - Load the LUMI module and modules should appear in your module view
  - EasyBuild-user module to install packages in your user space
  - Will use existing modules for dependencies if those are already on the system or in your personal/project stack
- EasyBuild built-in easyconfigs do not work well on LUMI, not even on LUMI-C
  - GNU-based toolchains: Would give problems with MPI (Open MPI)
  - Intel-based toolchains: Intel tools and AMD CPUs are a problematic cocktail
- Library of recipes that we made in the LUMI-EasyBuild-contrib GitHub repository
  - EasyBuild-user will find a copy on the system or in your installation
  - List of recipes in the LUMI Software Library

EasyBuild recipes - easyconfigs

LUMI

- Build recipe for an individual package = module
  - Relies on either a generic or a specific installation process provided by an easyblock
- Steps
  - Downloading and unpacking sources and applying patches
  - Typical configure – build – (test) – install process
  - Extensions mechanism for perl/python/R packages
  - Some simple checks
  - Creation of the module
- All have several parameters in the easyconfig file

The toolchain concept

LUMI

- A set of compiler, MPI implementation and basic math libraries
  - Simplified concept on LUMI as there is no hierarchy as on some other EasyBuild systems
- These are the cpeCray, cpeGNU, cpeAOCC and cpeAMD modules mentioned before!

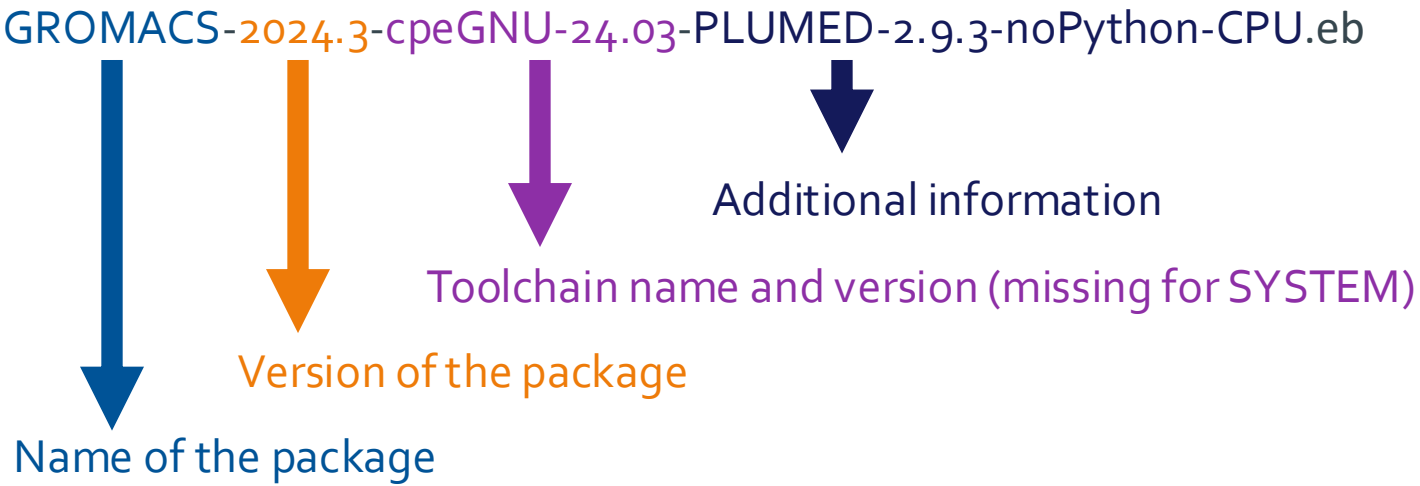
HPE Cray PE	LUMI toolchain	
PrgEnv-cray	cpeCray	Cray Compiling Environment
PrgEnv-gnu	cpeGNU	GNU C/C++ and Fortran
PrgEnv-aocc	cpeAOCC	AMD CPU compilers (not on LUMI-G)
PrgEnv-amd	cpeAMD	AMD ROCm GPU compilers (LUMI-G only)

The toolchain concept (2)

LUMI

- Special toolchain: SYSTEM to use the system compiler
  - Does not fully function in the same way as the other toolchains when it comes to dependency handling
  - Used on LUMI for CrayEnv and some packages with few dependencies
- It is not possible to load packages from different cpe toolchains at the same time
  - EasyBuild restriction, because mixing libraries compiled with different compilers does not always work
- Packages compiled with one cpe toolchain can be loaded together with packages compiled with the SYSTEM toolchain
  - But we do avoid mixing them when linking

easyconfig names and module names



Module: GROMACS/2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU

Installing
Step 1: Where to install

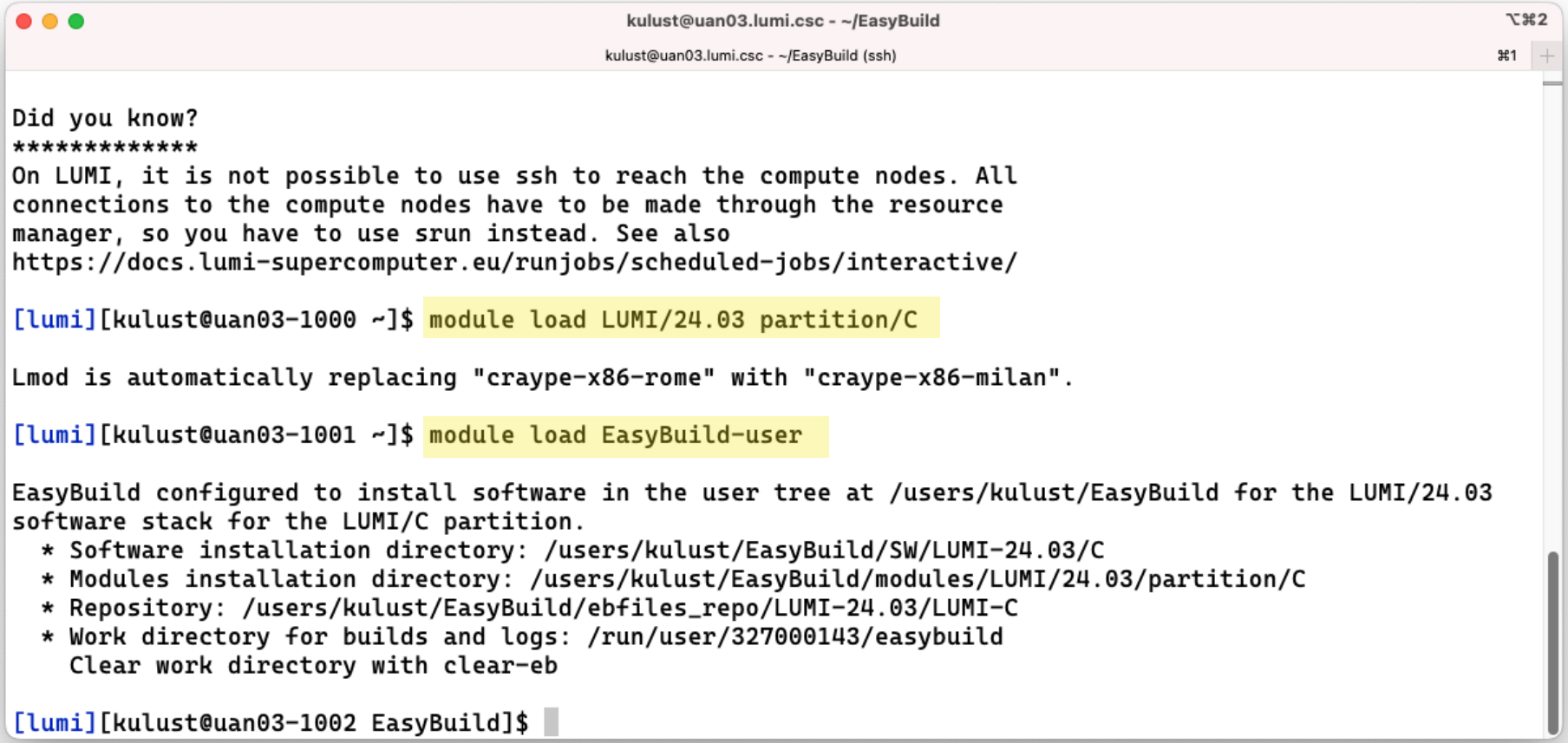
- Default location is `$HOME/EasyBuild`
- But better is to install in your project directory for the whole project
  - `export EBU_USER_PREFIX=/project/project_465000000/EasyBuild`
  - Set this *before* loading the `LUMI` module
  - All users of the software tree have to set this environment variable to use the software tree

Installing
Step 2: Configure the environment

LUMI

- Load the modules for the LUMI software stack and partition that you want to use. E.g.,
  ```
  module load LUMI/24.03 partition/C
  ```
- Load the EasyBuild-user module to make EasyBuild available and to configure it for installing software in the chosen stack and partition:
  ```
  module load EasyBuild-user
  ```
- In many cases, cross-compilation is possible by loading a different partition module than the one auto-loaded by LUMI
  - Though cross-compilation is sometimes problematic for GPU code

```
module load LUMI/24.03 partition/C
module load EasyBuild-user
```



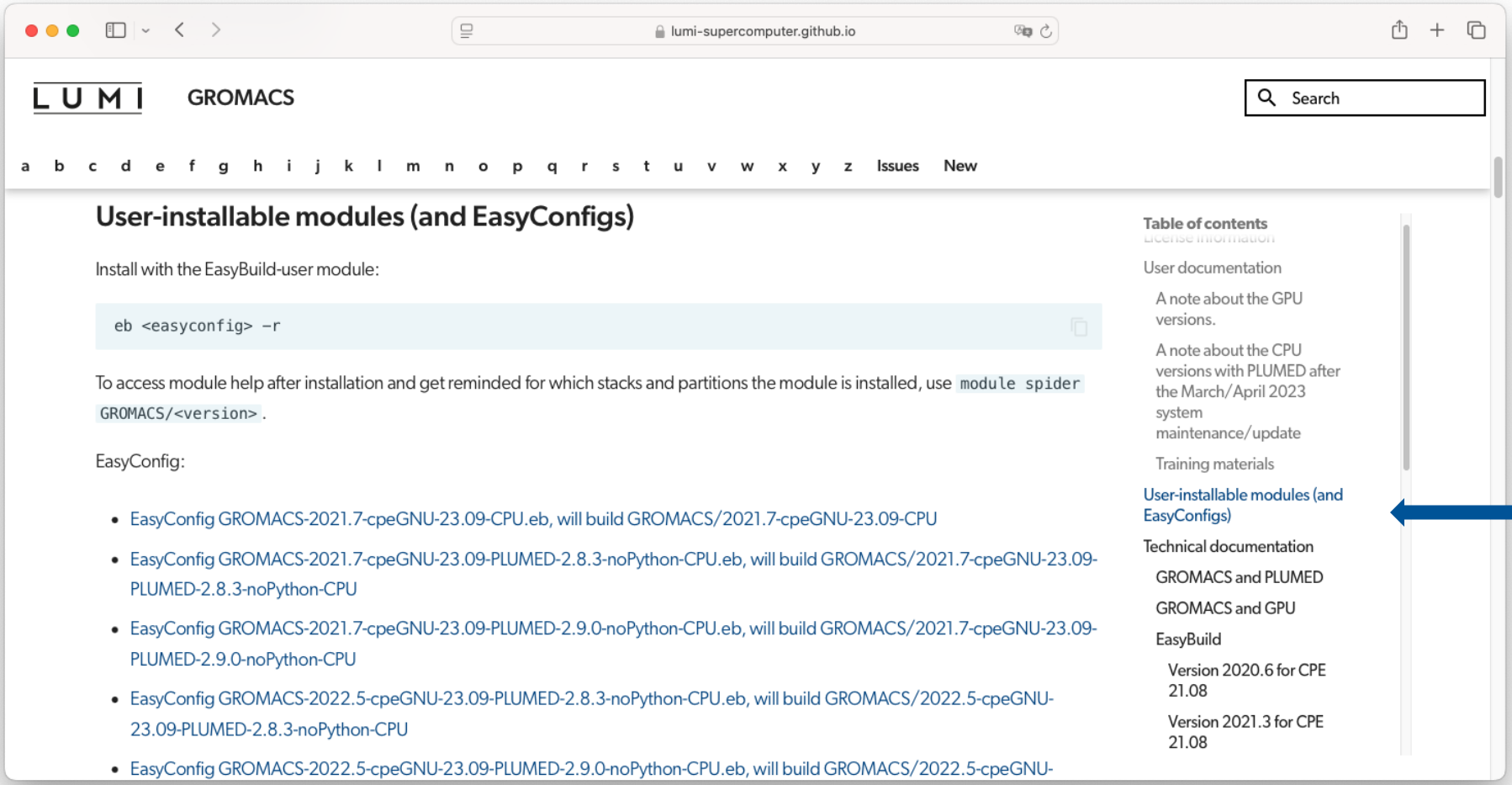
Installing
Step 3: Install the software

LUMI

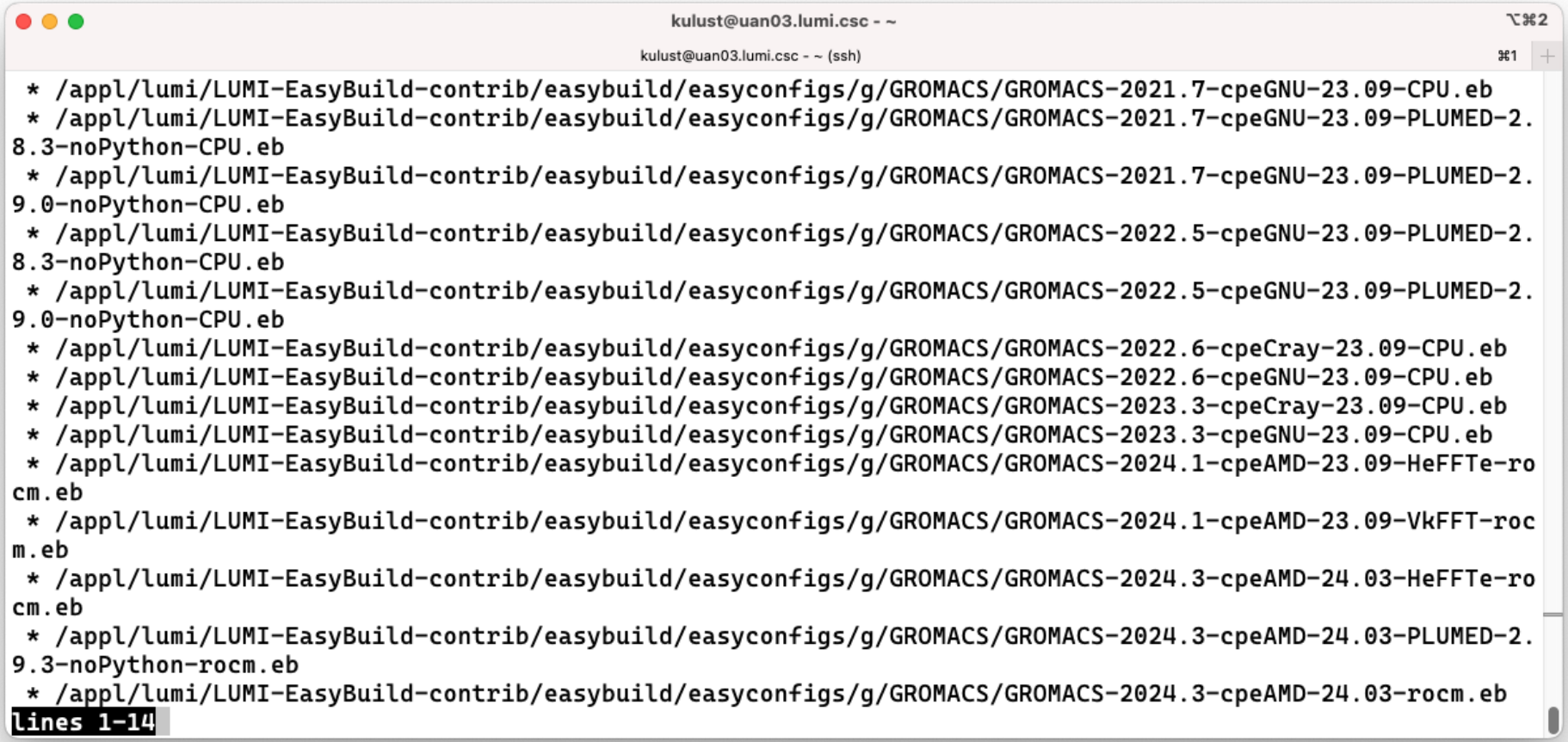
- Let's, e.g., install GROMACS
  - Search if GROMACS build recipes are available:
    - Search the LUMI Software Library that lists all available software through EasyBuild.
    - Or on the command line:

```
eb --search GROMACS
eb -S GROMACS
```

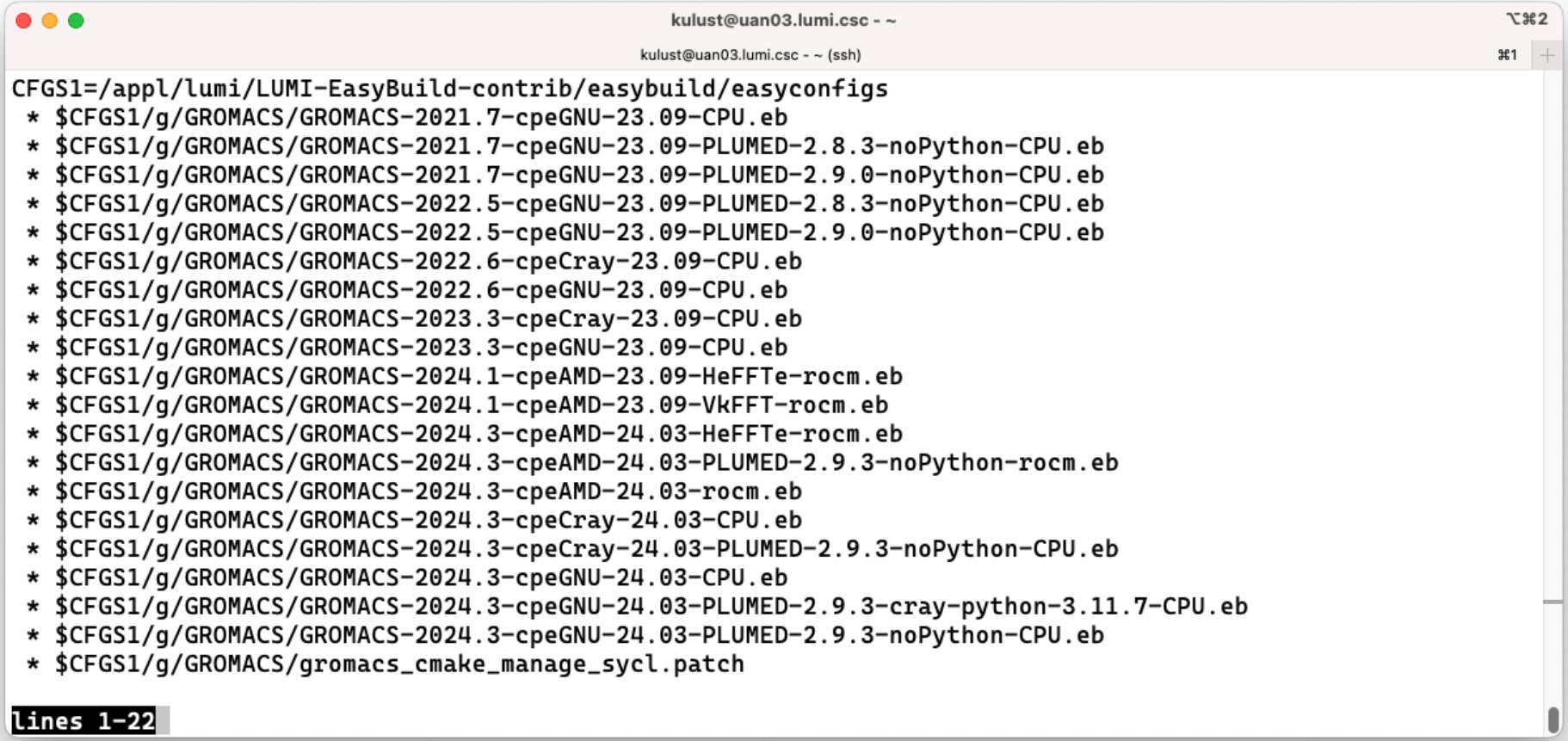
LUMI Software Library



eb --search GROMACS | less



eb -S GROMACS | less

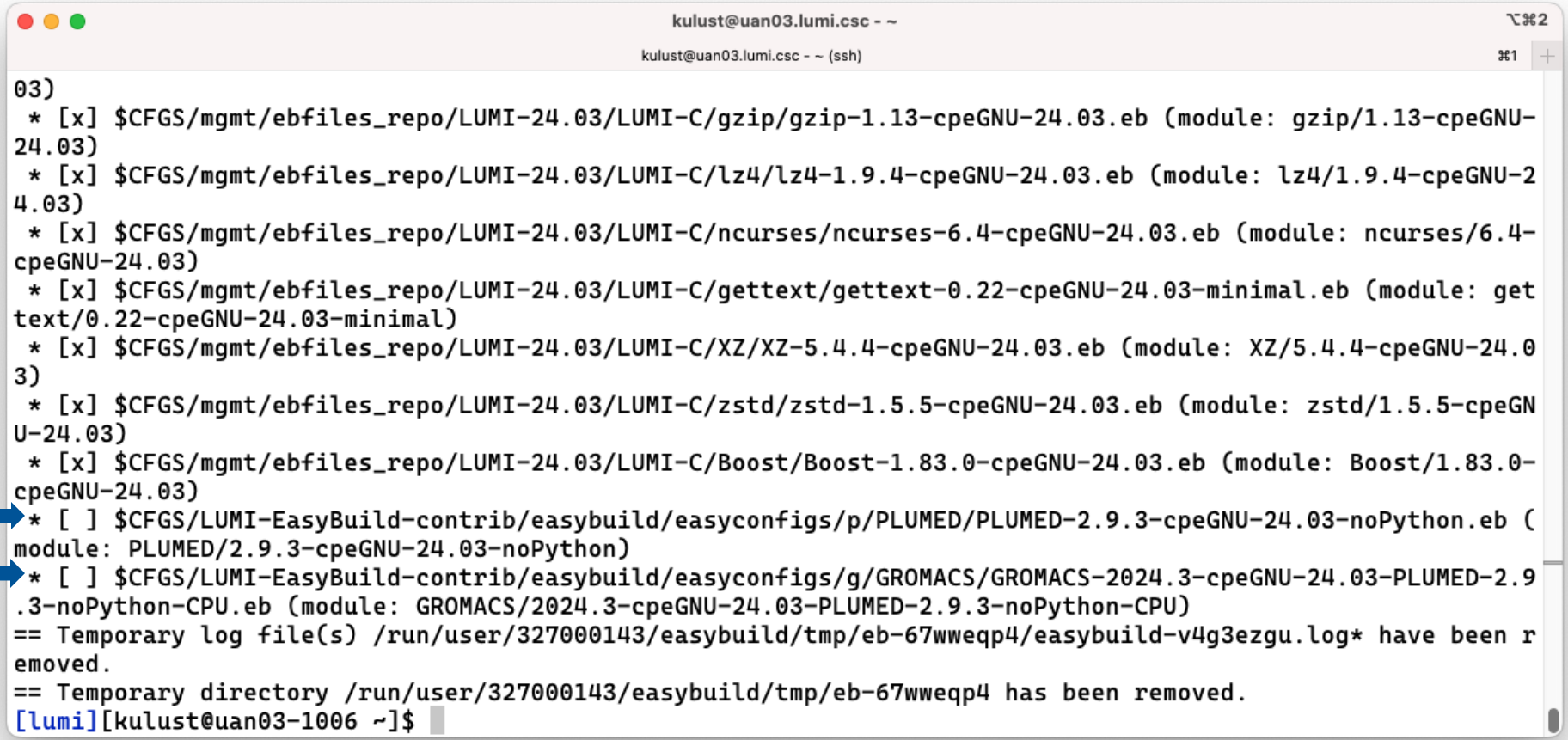


Installing
Step 3: Install the software

- Let's, e.g., install GROMACS
  - Search if GROMACS build recipes are available:
    - Search the LUMI Software Library that lists all available software through EasyBuild.
    - Or on the command line:
      ```
      eb --search GROMACS
      eb -S GROMACS
      ```
  - Let's take GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb:
    ```
    eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -D
    ```

eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -D

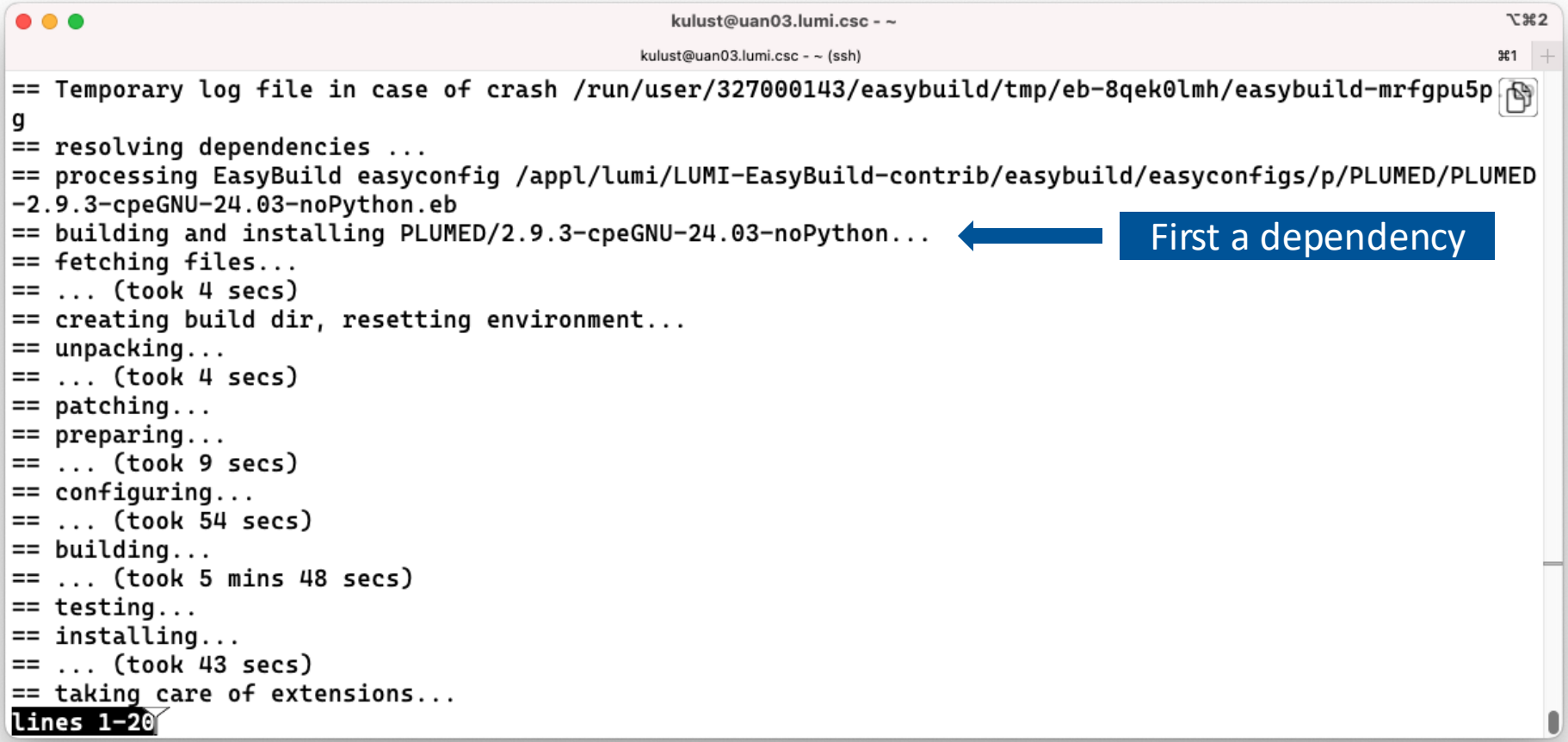
eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -D (2)



Installing
Step 3: Install the software

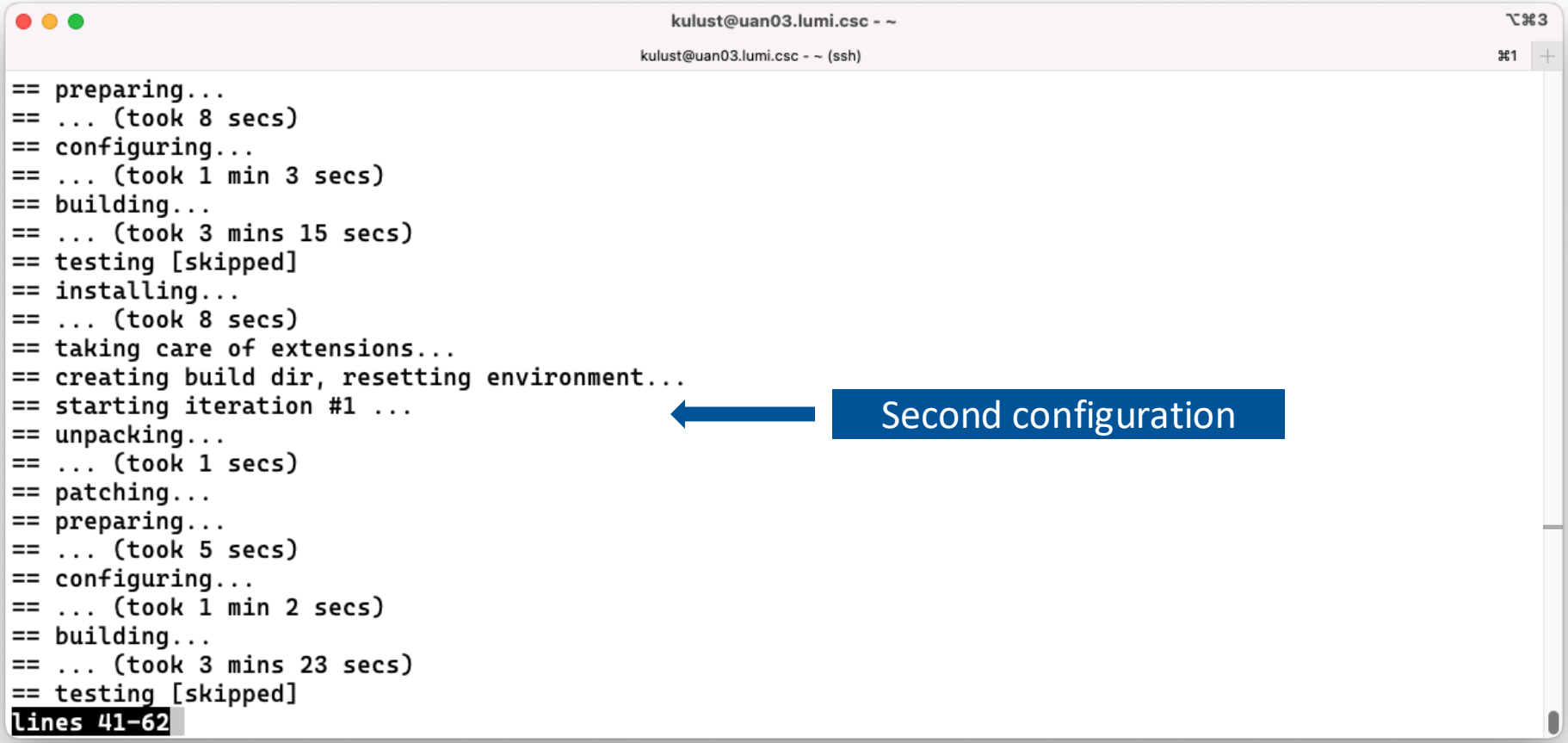
- Let's, e.g., install GROMACS
  - Search if GROMACS build recipes are available:
    - Search the LUMI Software Library that lists all available software through EasyBuild.
    - Or on the command line:
      ```
      eb --search GROMACS
      eb -S GROMACS
      ```
  - Let's take GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb:
    ```
    eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -D
    eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -r
    ```

eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -r

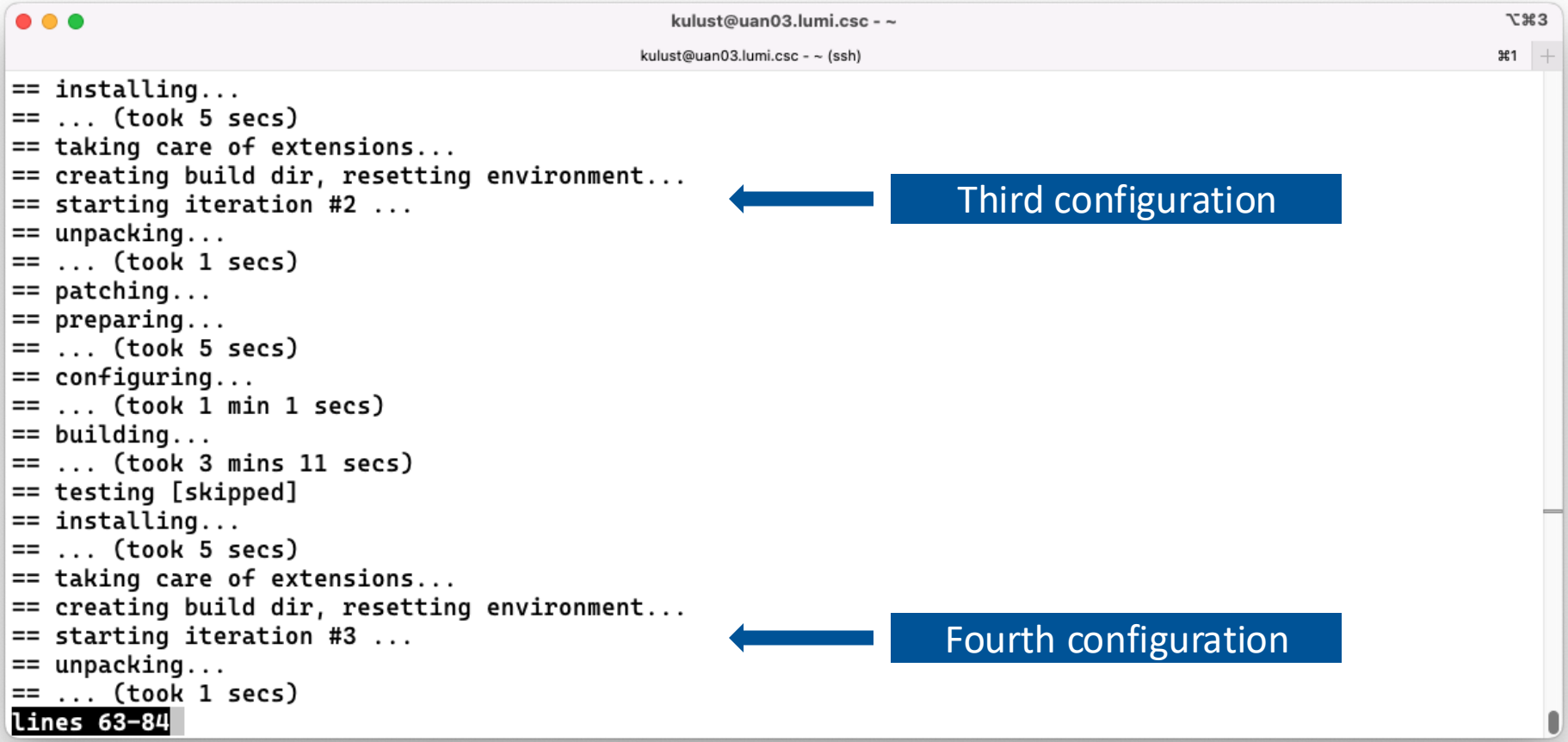


eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -r (2)

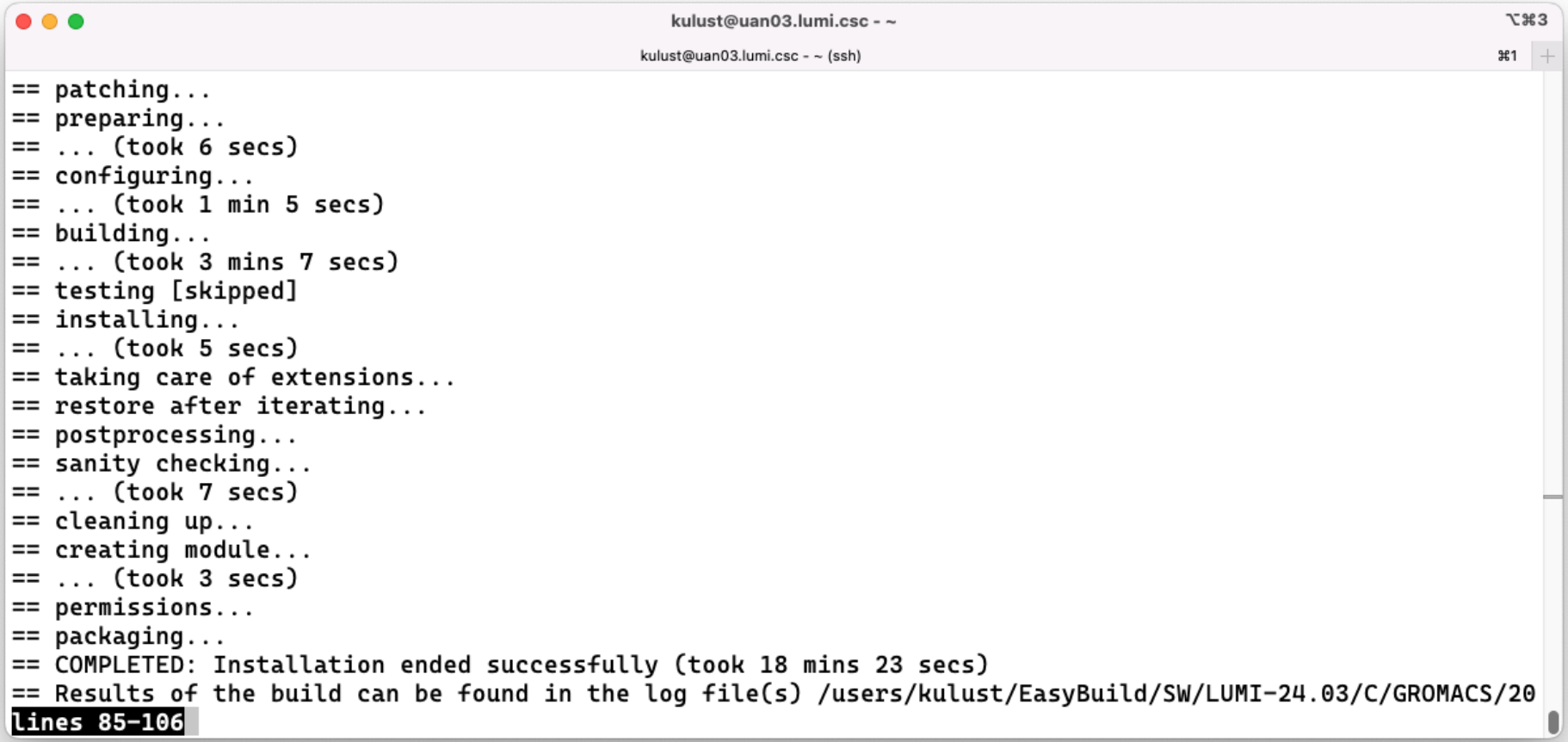
eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -r(3)



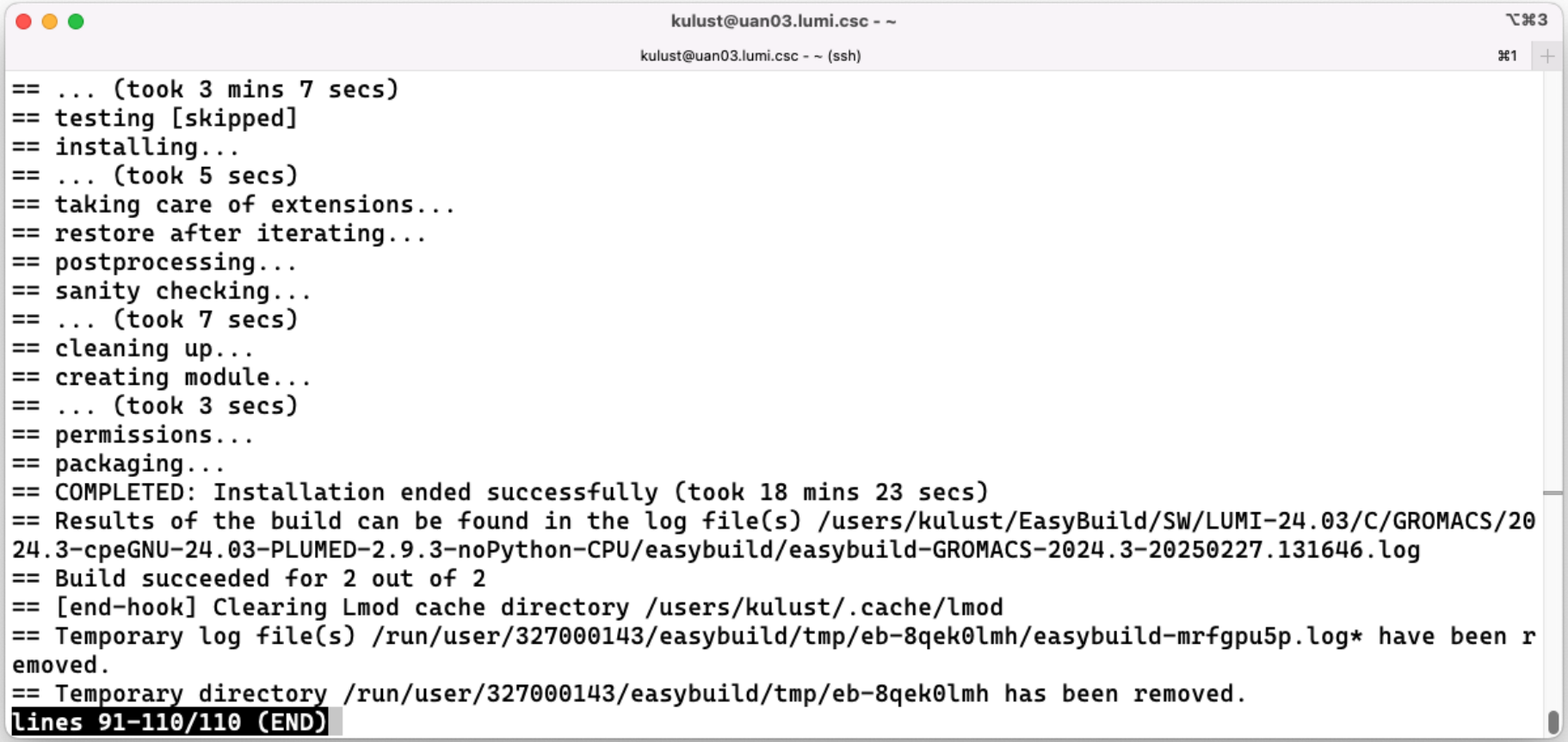
eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -r (4)



eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -r (5)



eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -r (6)



Installing
Step 3: Install the software

LUMI

- Let's, e.g., install GROMACS
  - Search if GROMACS build recipes are available:
    - Search the LUMI Software Library that lists all available software through EasyBuild.
    - Or on the command line:
      ```
      eb --search GROMACS
      eb -S GROMACS
      ```
  - Let's take GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb:
    ```
    eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -D
    eb GROMACS-2024.3-cpeGNU-24.03-PLUMED-2.9.3-noPython-CPU.eb -r
    ```
- Now the module should be available
  ```
  module avail GROMACS
  ```

Installing
Step 3: Install the software - Note

- Installing this way is 99% equivalent to an installation in the central software tree. The application is compiled in exactly the same way as we would do and is served from Lustre in both cases.
  - But you are in control of updates.
- Note: EasyBuild clears the Lmod user cache so in principle newly installed modules should show up without problems after installation.
  - We've seen rare cases where internal Lmod data structures were corrupt and logging out and in again was needed.
- To manually remove the cache: Remove `$HOME/.cache/lmod`

```
rm -rf $HOME/.cache/lmod
```

More advanced work

LUMI

- You can also install some EasyBuild recipes that you got from support and are in the current directory (preferably one without subdirectories):
  `eb my_recipe.eb -r .`
  - Note the dot after the `-r` to tell EasyBuild to also look for dependencies in the current directory (and its subdirectories)
- In some cases you will have to download the sources by hand, e.g., for VASP, which is then at the same time a way for us to ensure that you have a license for VASP. E.g.,
  - `eb --search VASP`
  - Then from the directory with the VASP sources:
    `eb VASP-6.5.0-cpeGNU-24.03-build02.eb -r .`

More advanced work (2): Repositories

LUMI

- It is possible to have your own clone of the LUMI-EasyBuild-contrib repo in your `$EBU_USER_PREFIX` subdirectory if you want the latest and greatest before it is in the centrally maintained repository
  - ```
    cd $EBU_USER_PREFIX
    git clone https://github.com/Lumi-supercomputer/LUMI-EasyBuild-contrib.git
    ```
- It is also possible to maintain your own repo
  - The directory should be `$EBU_USER_PREFIX/UserRepo` (but of course on GitHub the repository can have a different name)
  - Structure should be compatible with EasyBuild: easyconfig files go in `$EBU_USER_PREFIX/UserRepo/easybuild/easyconfigs`
    ```

More advanced work (3): Reproducibility

LUMI

- EasyBuild will keep a copy of the sources in `$EBU_USER_PREFIX/sources`
- EasyBuild also keeps copies of all installed easyconfig files in two locations:
  - In `$EBU_USER_PREFIX/ebfiles_repo`
    - And note that EasyBuild will use this version if you try to reinstall and did not delete this version first!
    - This ensures that the information that EasyBuild has about the installed application is compatible with what's in the module files
  - With the installed software (in `$EBU_USER_PREFIX/SW`) in a subdirectory called easybuild
    This is meant to have all information about how EasyBuild installed the application and to help in reproducing

EasyBuild tips&tricks

LUMI

- **Updating version:** Often some trivial changes in the EasyConfig (`.eb`) file
  - Checksums may be annoying: Use `--ignore-checksums` with the `eb` command
- **Updating to a new toolchain:**
  - Be careful, it is more than changing one number
  - Versions of preinstalled dependencies should be changed and EasyConfig files of other dependencies also checked
- LUMI Software Library at lumi-supercomputer.github.io/LUMI-EasyBuild-docs
  - For most packages, pointers to the license
  - User documentation gives info about the use of the package, or restrictions
  - Technical documentation aimed at users who want more information about how we build the package

EasyBuild training for advanced users and developers

LUMI

- EasyBuild web site: easybuild.io
- Generic EasyBuild training materials on tutorial.easybuild.io.
- Training for CSC and local support organisations: Most up-to-date version of the training materials on lumi-supercomputer.github.io/easybuild-tutorial.
- Possibly a new training for LUMI users in 2026


LUMI
Questions?