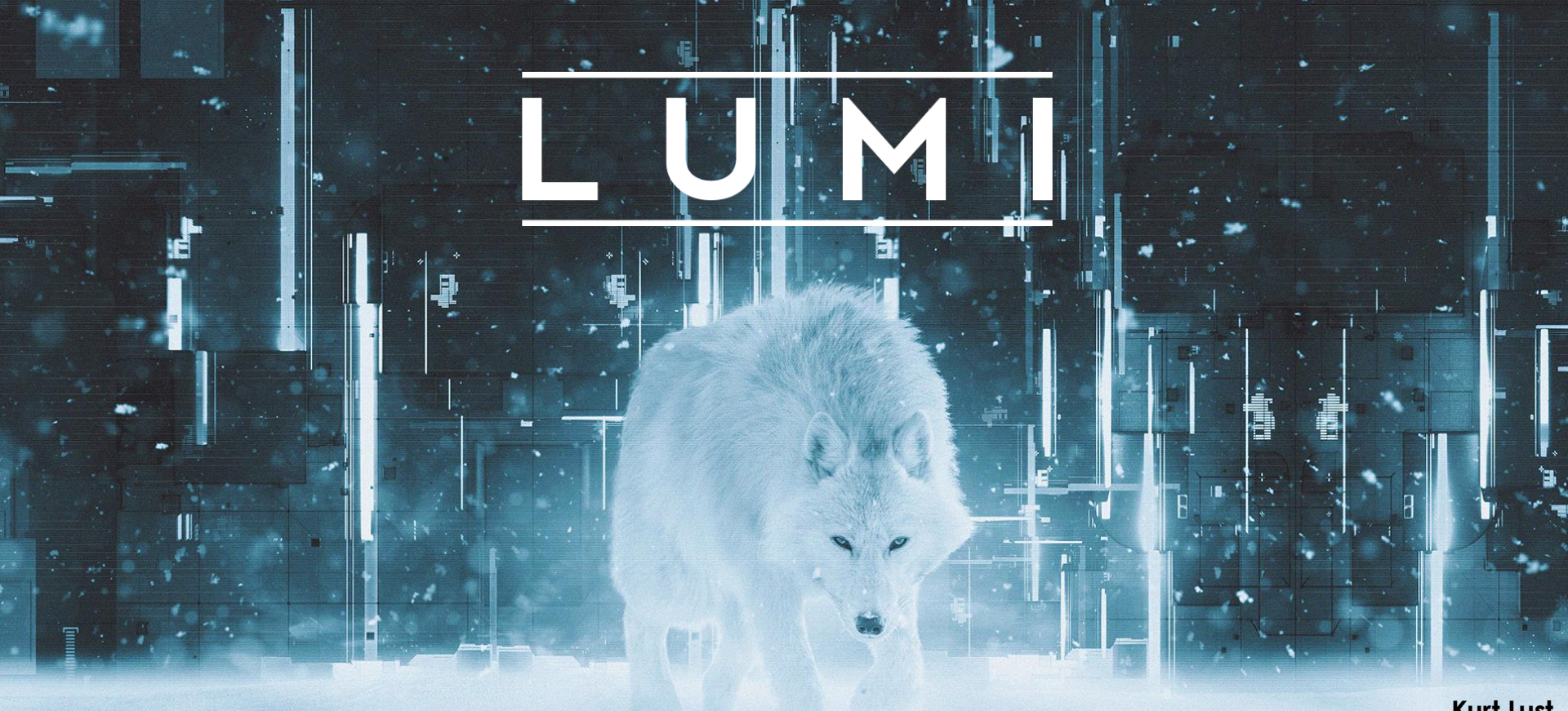


Containers on LUMI-C and LUMI-G

Kurt Lust
LUMI User Support Team (LUST)
University of Antwerp

November 2026

Containers

LUMI

This is about containers on LUMI-C and LUMI-G, not LUMI-K!

- What can they do and what can't they do?
- Getting containers onto LUMI
- Running containers on LUMI
- Enhancements to the LUMI environment to help you
- Strategies for extending existing containers

- But remember: LUMI is an HPC infrastructure, not a container cloud!
  - HPC has is own container runtimes specifically for an HPC environment

What do containers not provide?

LUMI

- **Full reproducibility of your science** is a myth
  - Only reproducibility of part of the software stack, not of the results
- **Performance portability**:
  - A container built from sources on one CPU will not be optimal for another one.
  - Containers built from downloaded binaries may not exploit all architectural features of the CPU.
  - No support for the LUMI interconnect may lead to fall-back to a slower protocol that works
- **Simply portability**: Not every container prepared on your Ubuntu or CentOS cluster or workstation will work on LUMI.
  - Containers that rely on certain hardware, drivers/kernel modules and/or kernel versions may fail.
  - Problem cases: GPU (driver version) and sometimes high-performance networking (MPI, RCCL, …)

But what can they then do on LUMI?

LUMI

- **Containers are a sofware management instrument** to create a contained software installation
- **Storage manageability:** Lower pressure on the filesystems (for software frameworks that access hundreds of thousands of small files) for better I/O performance and management of your disk file quota.
  - E.g., conda installations are not appreciated straight on the Lustre file system
- **Software installation:** Can be a way to install software with an installation process that is not aware of multi-user HPC systems and is too complicated to recompile.
  - E.g., GUI applications that need a fat library stack
  - E.g., experiment with software that needs a newer version of ROCm, though with limitations
- **Isolation:** More important for services; often a pain instead
  - But still helps avoiding picking up the wrong libraries
- **But note:** You're the system administrator of your container, not LUST!

Storage manageability: Python

LUMI

Profiling can reveal things we forget

LUMI

- We were experimenting with Cray performance tools profiling the LUMI MNIST image example
- Using a venv for pytorch torchvision torchaudio with rocm 6.0 support
- Run on 4 nodes 32 GPUs
- Looking at the profile most of the I/O was in python files and read/open/dlopen
- Moved to putting the venv into the container:
  Runtime changed from 100s to 52s

Managing containers

LUMI

- Supported runtimes
  - Not all container runtimes are a good match with HPC systems
    - No elevated privileges for security reasons
    - No user namespaces on LUMI because of security issues in combination with networked filesystems
  - Docker is **NOT** directly available in the user environment (and will never be)
  - Singularity Community Edition is natively available (as a system command) on the login and compute nodes

Managing containers (2)

LUMI

- But you can convert docker containers to singularity: Pulling containers
  - DockerHub and other registries (example: Julia container)
    `singularity pull docker://julia`
  - Singularity uses a flat (single) sif file for storing the container and the pull command makes the conversion
  - Be carefull: cache in `.singularity` dir can easily exhaust your storage quota for larger images
    - May want to set `SINGULARITY_CACHEDIR` to move the cache

singularity pull docker://julia

LUMI

kulust@uan04.lumi.csc - ~ ⌥⌘2

kulust@uan04.lumi.csc - ~ (ssh) ⌘1

```
[lumi][kulust@uan04-1007 ~]$ singularity pull docker://julia
INFO:    Converting OCI blobs to SIF format
INFO:    Starting build...
INFO:    Fetching OCI image...
6.0MiB / 6.0MiB [==========================================================================] 100 % 64.7 MiB/s 0s
28.5MiB / 28.5MiB [========================================================================] 100 % 64.7 MiB/s 0s
292.6MiB / 292.6MiB [======================================================================] 100 % 64.7 MiB/s 0s
INFO:    Extracting OCI image...
2026/10/30 15:28:42  warn rootless{usr/local/julia/lib/julia/libLLVM-20jl.so} ignoring (usually) harmless
EPERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:42  warn rootless{usr/local/julia/lib/julia/libLLVM.so} ignoring (usually) harmless EPERM
 on setxattr "user.rootlesscontainers"
2026/10/30 15:28:44  warn rootless{usr/local/julia/lib/julia/libamd.so} ignoring (usually) harmless EPERM
on setxattr "user.rootlesscontainers"
2026/10/30 15:28:44  warn rootless{usr/local/julia/lib/julia/libamd.so.3} ignoring (usually) harmless EPER
M on setxattr "user.rootlesscontainers"
2026/10/30 15:28:44  warn rootless{usr/local/julia/lib/julia/libatomic.so} ignoring (usually) harmless EPE
RM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:44  warn rootless{usr/local/julia/lib/julia/libatomic.so.1} ignoring (usually) harmless E
PERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:44  warn rootless{usr/local/julia/lib/julia/libblastrampoline.so} ignoring (usually) harm
less EPERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:44  warn rootless{usr/local/julia/lib/julia/libblastrampoline.so.5.15.0} ignoring (usuall
```

singularity pull docker://julia

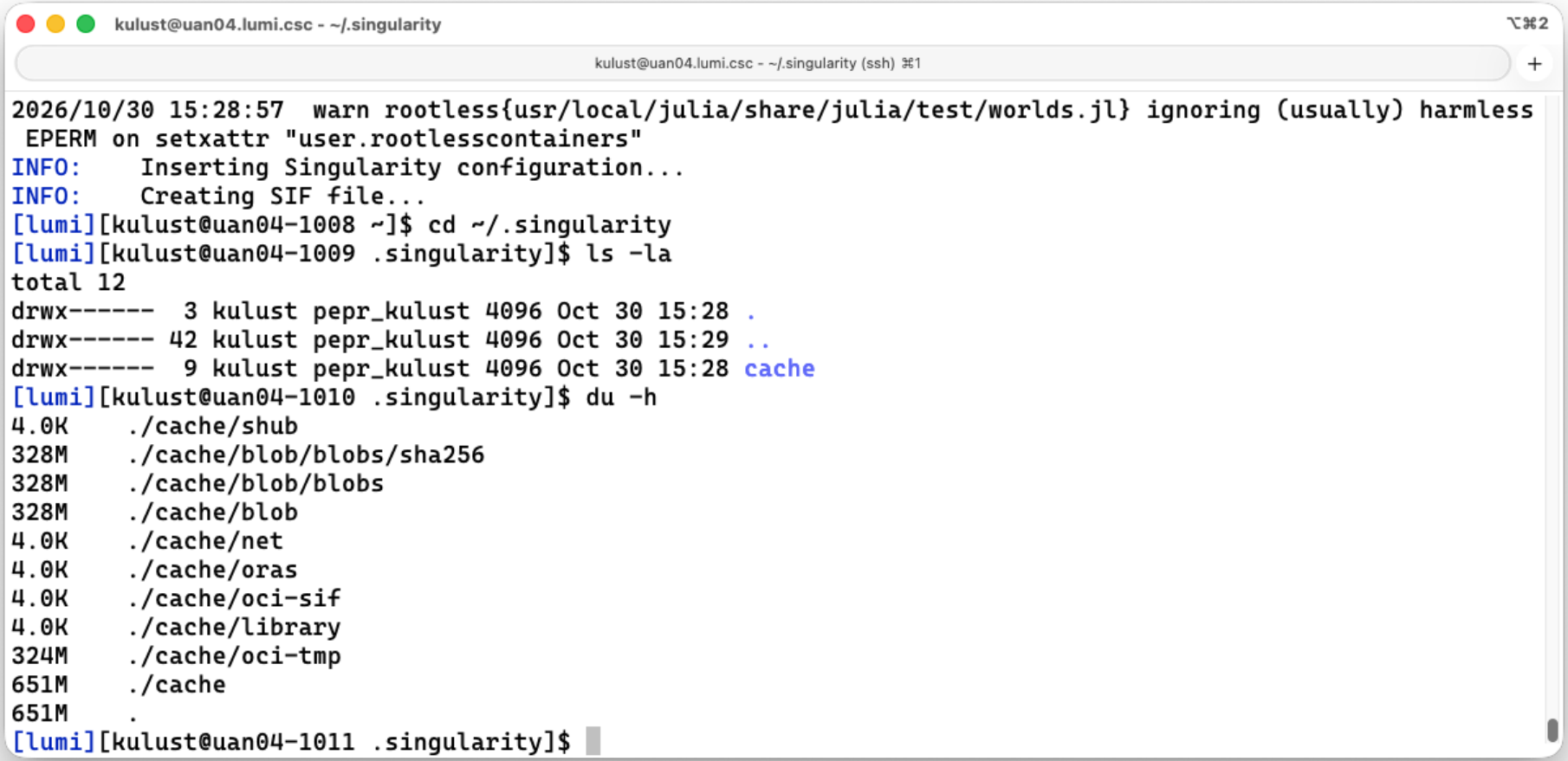
LUMI

kulust@uan04.lumi.csc - ~ ⌥⌘2

kulust@uan04.lumi.csc - ~ (ssh) ⌘1

```
2026/10/30 15:28:57  warn rootless{usr/local/julia/share/julia/test/trimming/juliac-trim-stdlib.jl} ignori
ng (usually) harmless EPERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:57  warn rootless{usr/local/julia/share/julia/test/trimming/libsimple.jl} ignoring (usual
ly) harmless EPERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:57  warn rootless{usr/local/julia/share/julia/test/trimming/trimmability.jl} ignoring (us
ually) harmless EPERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:57  warn rootless{usr/local/julia/share/julia/test/trimming/trimming.jl} ignoring (usuall
y) harmless EPERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:57  warn rootless{usr/local/julia/share/julia/test/triplequote.jl} ignoring (usually) har
mless EPERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:57  warn rootless{usr/local/julia/share/julia/test/tuple.jl} ignoring (usually) harmless
EPERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:57  warn rootless{usr/local/julia/share/julia/test/unicode/utf8.jl} ignoring (usually) ha
rmless EPERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:57  warn rootless{usr/local/julia/share/julia/test/vecelement.jl} ignoring (usually) harm
less EPERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:57  warn rootless{usr/local/julia/share/julia/test/version.jl} ignoring (usually) harmles
s EPERM on setxattr "user.rootlesscontainers"
2026/10/30 15:28:57  warn rootless{usr/local/julia/share/julia/test/worlds.jl} ignoring (usually) harmless
 EPERM on setxattr "user.rootlesscontainers"
INFO:    Inserting Singularity configuration...
INFO:    Creating SIF file...
[lumi][kulust@uan04-1008 ~]$
```

singularity pull docker://julia



Managing containers (3)

LUMI

- Building containers
  - Support for building containers is very limited on LUMI: No elevated privileges but also no user namespaces and no fakeroot.
  - One option is to pull or copy containers from outside.
  - We can support `proot` though and SingularityCE has a build process based on this. With that, singularity can build from existing (base) container in some cases.
    - Build type called "Unprivileged proot builds" in the Singularity CE manual.
    - Needs `proot` from the `PRoot` module in CrayEnv or the LUMI stacks.
  - We provide some base images adapted for LUMI.

Interacting with containers

LUMI

- Accessing a container with the shell command

```
singularity shell container.sif
```

singularity shell julia_latest.sif

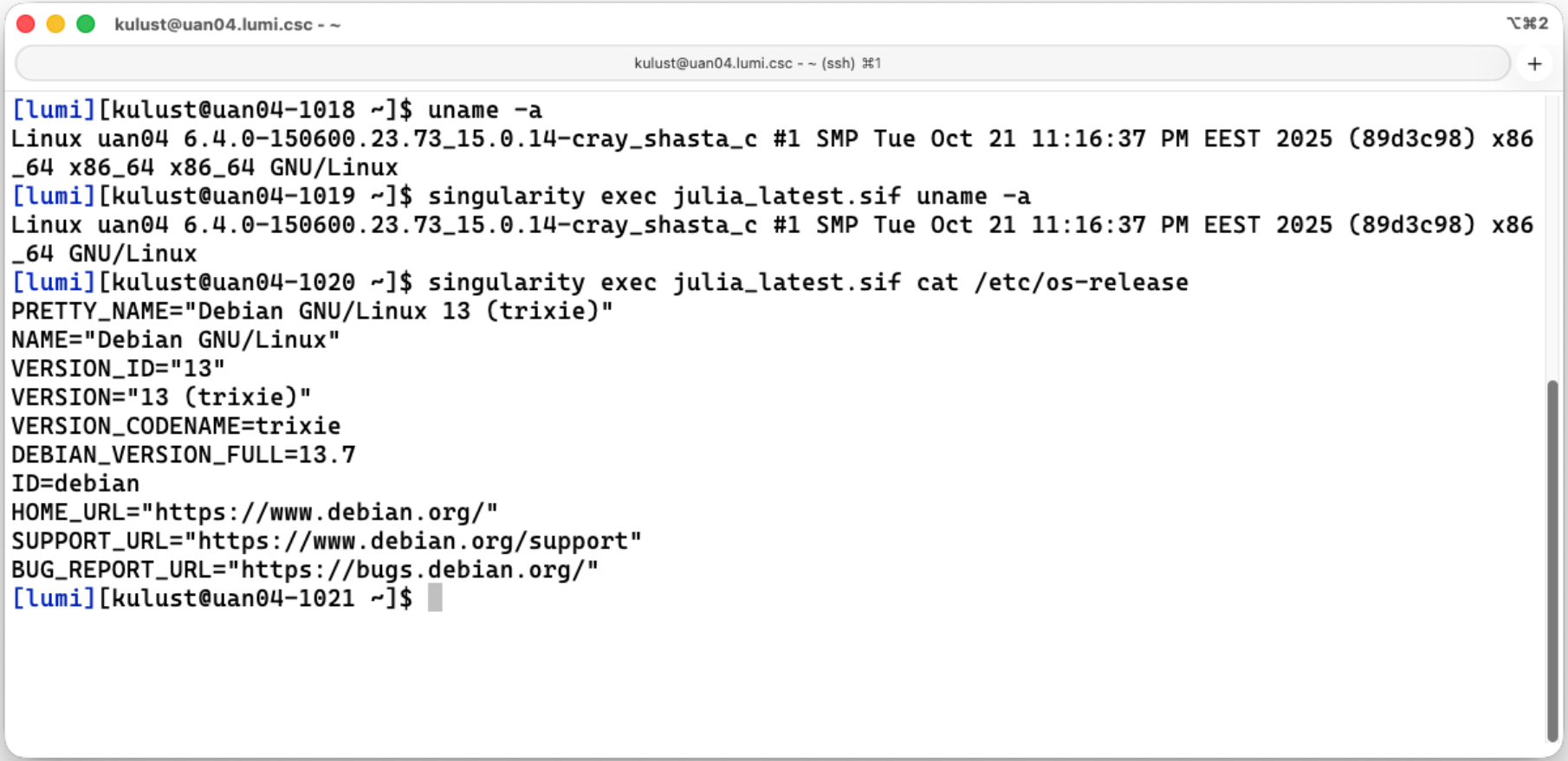


Interacting with containers

LUMI

- Accessing a container with the shell command
  `singularity shell container.sif`
- Executing a command in the container with exec
  `singularity exec container.sif uname -a`

singularity exec julia_latest.sif uname -a

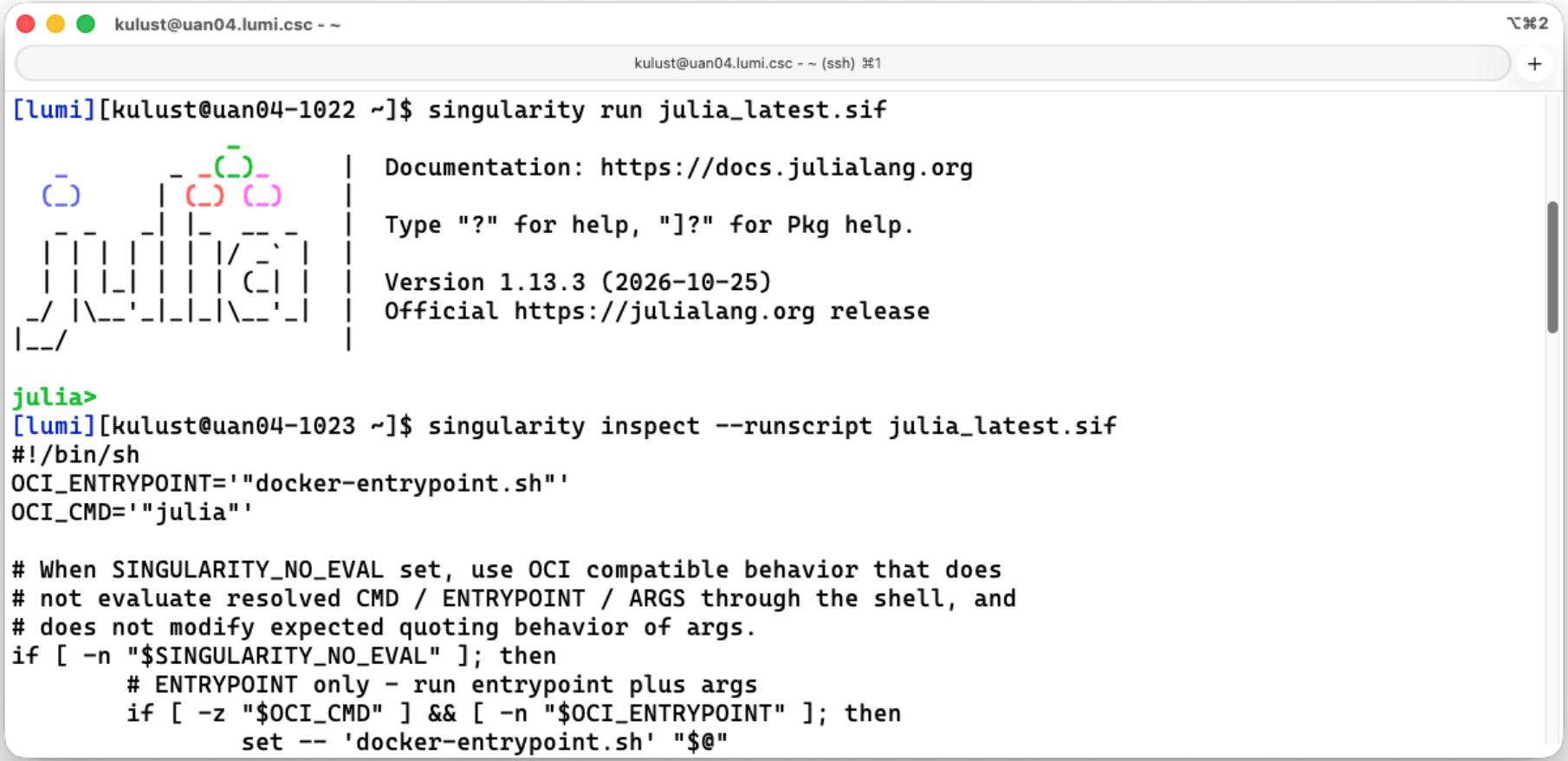


Interacting with containers

LUMI

- Accessing a container with the shell command
  `singularity shell container.sif`
- Executing a command in the container with exec
  `singularity exec container.sif uname -a`
- "Running" a container
  `singularity run container.sif`
  will usually execute a default command and pass its arguments to that command
- Inspecting run definition script
  `singularity inspect --runscript container.sif`

```
singularity run julia_latest.sif
singularity inspect -runscript julia_latest.sif
```



Interacting with containers

LUMI

- Accessing a container with the shell command
  `singularity shell container.sif`
- Executing a command in the container with exec
  `singularity exec container.sif uname -a`
- "Running" a container
  `singularity run container.sif`
  - Special on the LAIF containers: Simply execute the arguments as a command
- Inspecting run definition script
  `singularity inspect --runscript container.sif`
- Accessing host filesystem with bind mounts
  - Singularity will mount `$HOME`, `/tmp`, `/proc`, `/sys`, `/dev` into container by default
  - Use `--bind src1:dest1,src2:dest2` or the `SINGULARITY_BIND(PATH)` environment variable to mount other host directories (like `/project` or `/appl`). On LUMI you need `--bind /pfs,/scratch,/projappl,/project,/flash`

Running containers on LUMI

LUMI

- Use SLURM to run containers on compute nodes
- Use srun to execute MPI containers

```
srun singularity exec --bind ${BIND_ARGS} \
${CONTAINER_PATH} my_mpi_binary ${APP_PARAMS}
```

- **Be aware your container must be compatible with Cray MPI** (MPICH ABI compatible) or use the proper libfabric library with libcxi for good performance
  - Configure suggestion: see next slide
- Open MPI based containers may need workarounds and are not well supported on LUMI at the moment (and even more problematic for the GPU)

Environment enhancements (1)

LUMI

- LUMI specific tools for container interaction provided as modules
- **singularity-bindings/system** (available via easyconfig)
  - Sets the environment to use Cray MPICH provided outside the container
  - Requires a LUMI software stack
  - Use EasyBuild-user module and eb --search singularity-bindings to find the easyconfig or copy from our LUMI Software Library web site
  - Provides basic bind mounts for using the host MPI in the container setting `SINGULARITY_BIND` and `SINGULARITY_LD_LIBRARY_PATH`
  - May need customisation for your specific container

Environment enhancements (2)

LUMI

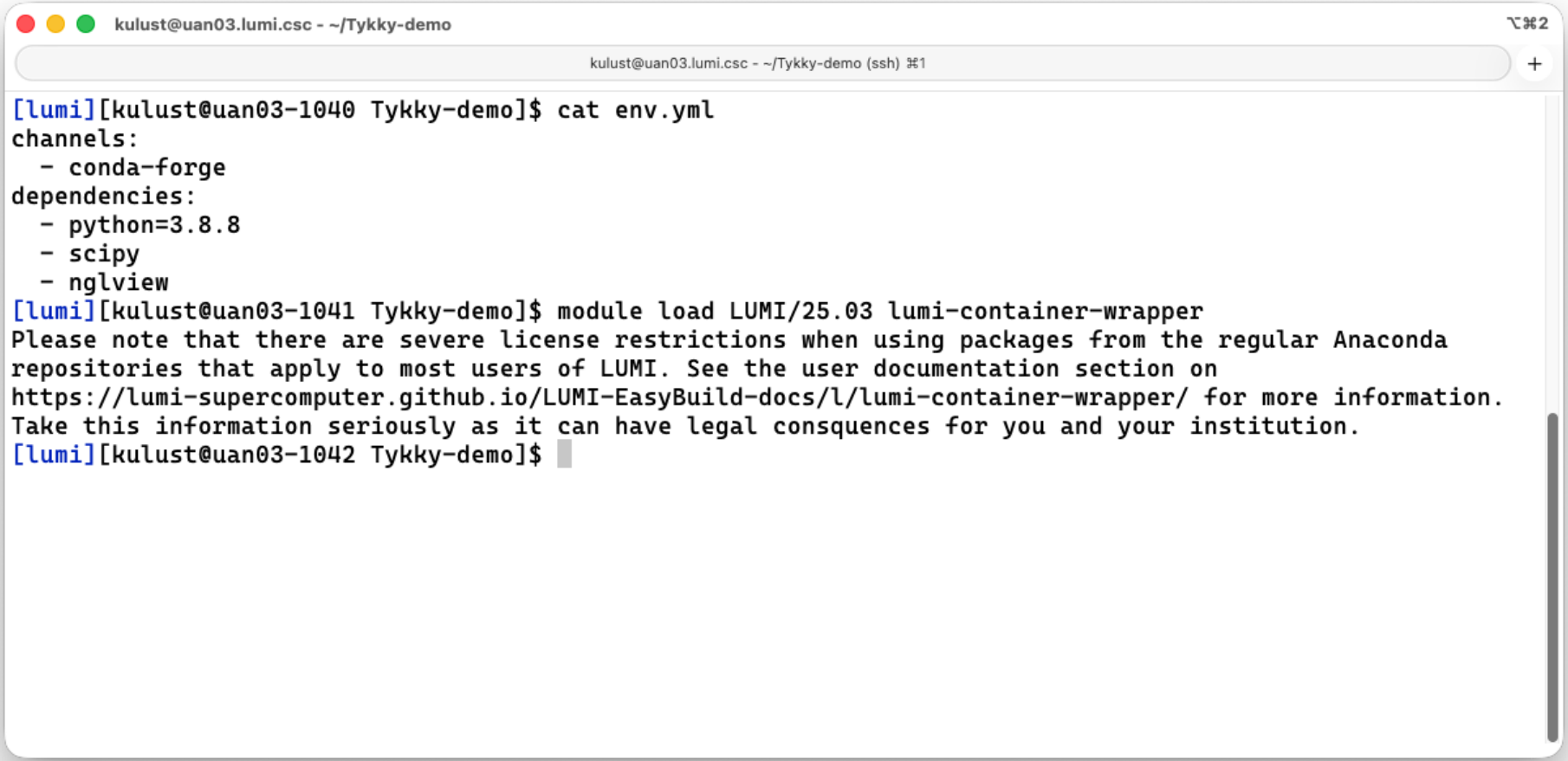
- **lumi-aif-singularity-bindings** (available via `Local-LAIF` module or `/appl/local/laifs/modules`)
  - Bindings for the AI containers provided by the LUMI AI Factory
  - Provide all your filespaces and a directory needed for Slurm support
  - Sets an environment variable so that its MPI can talk to Slurm
- **singularity-AI-bindings** (easyconfig or `/appl/local/containers/ai-modules`)
  - Bindings for some of the AI containers that LUST provided
  - But not a generic binding that will work for all containers!

Environment enhancements (3) Containerising tools

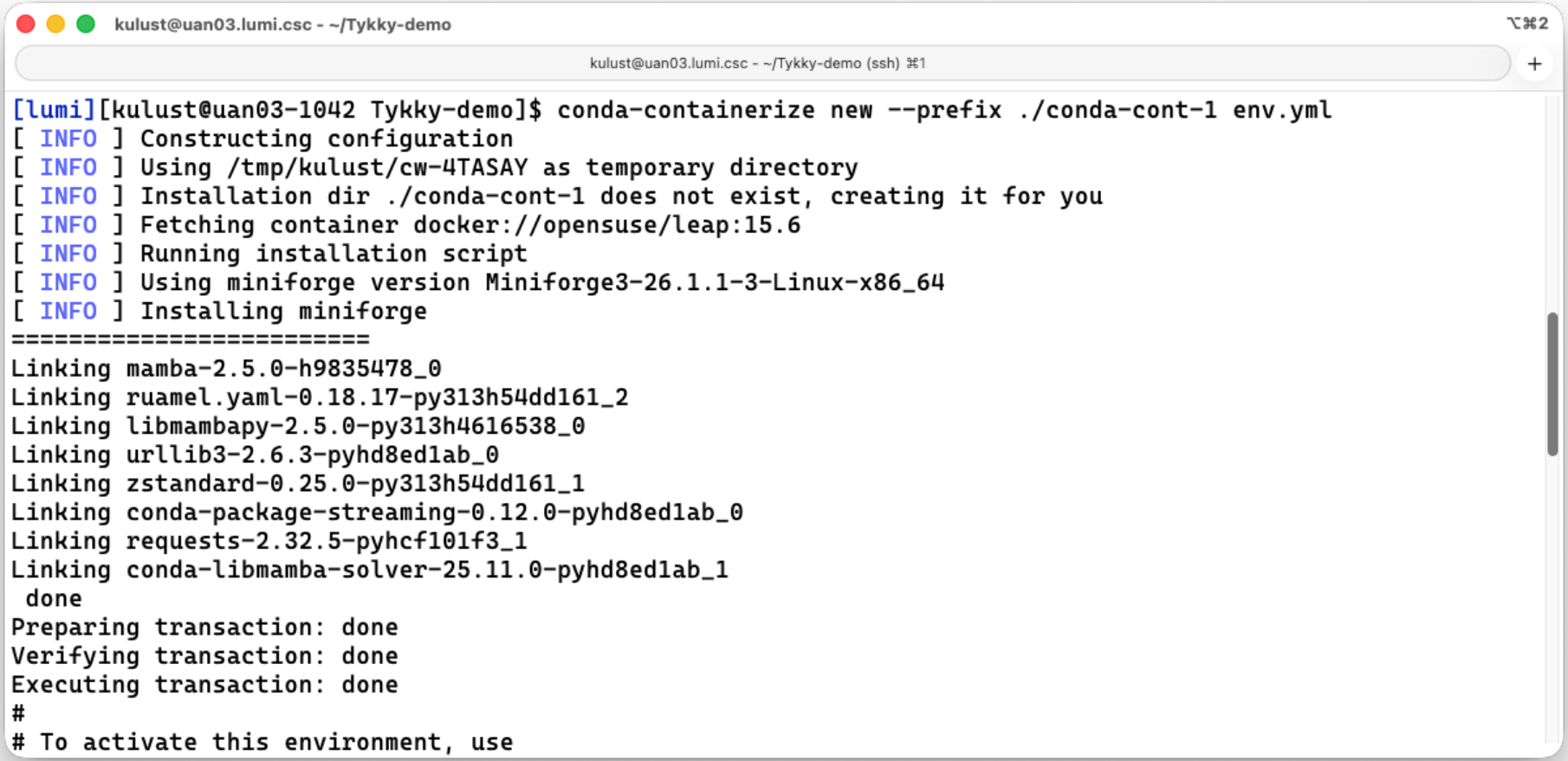
LUMI

- **cotainr** (LUMI and CrayEnv software stacks)
  - A tool to pack conda installations in a singularity container
  - Use the singularity commands as shown on earlier slides to run
- **lumi-container-wrapper** (LUMI and CrayEnv software stacks)
  - Supports conda and pip environments
  - With pip: Python provided by the cray-python module (so there is an optimised NumPy etc.)
  - Software installation in two parts: a base container and a SquashFS file which is mounted in that container with the conda/pip environment
  - Provides wrappers to encapsulate your custom environment in a container (so you don't use singularity commands directly)
    - Can even create wrappers for commands in an existing container
  - Still helps with quota on the number of files in your project and I/O performance

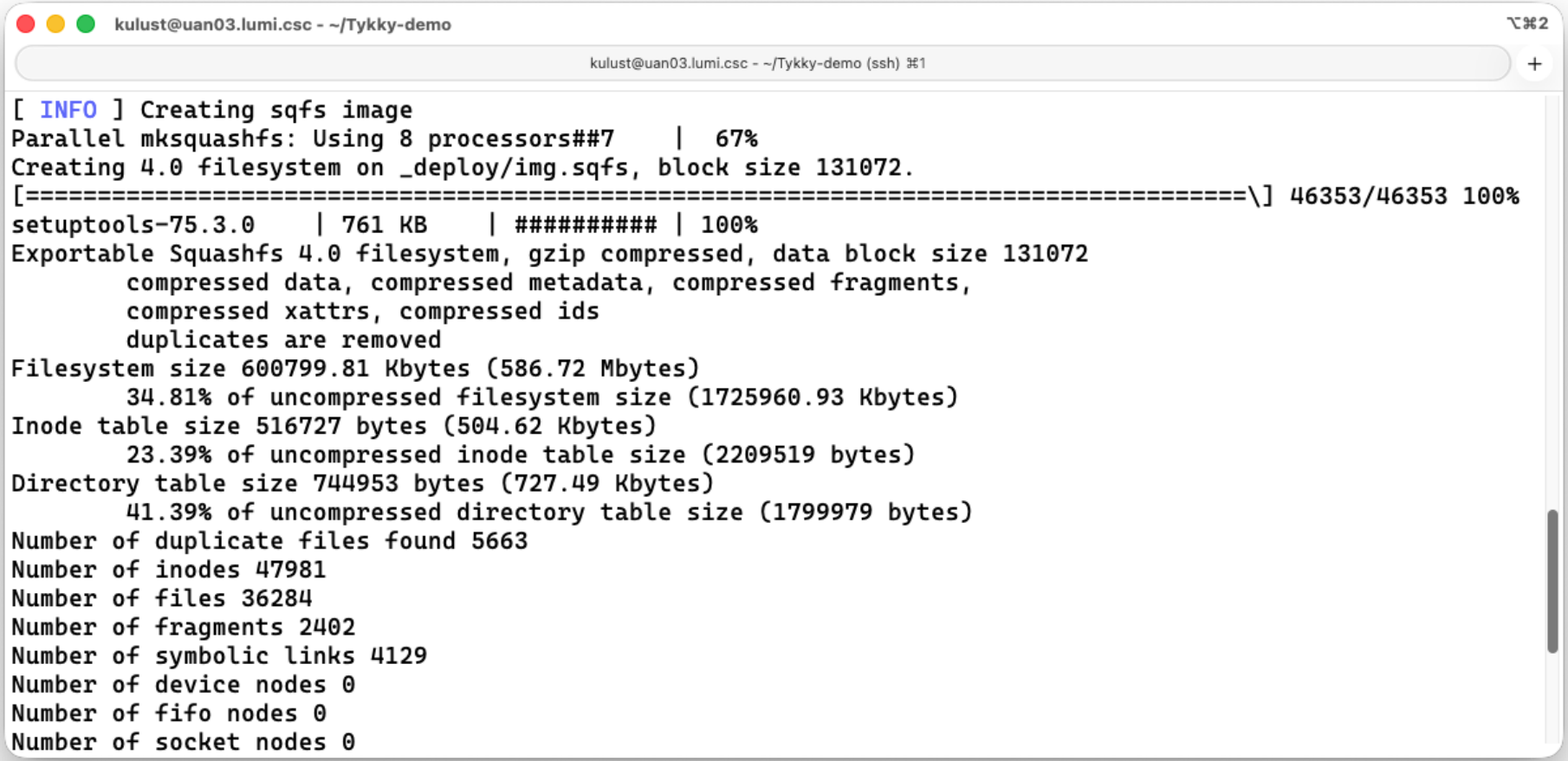
lumi-container-wrapper (1)



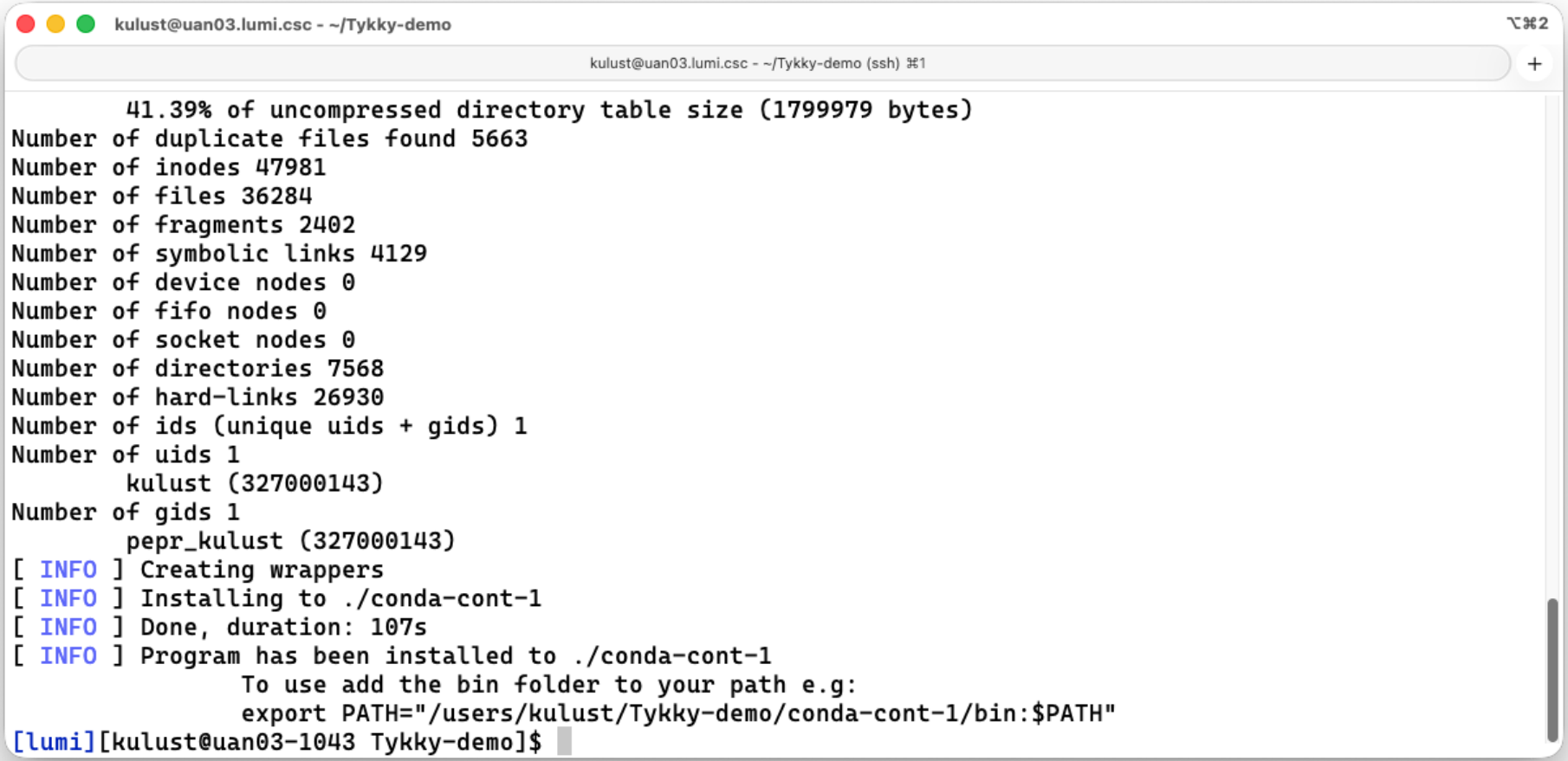
lumi-container-wrapper (2)



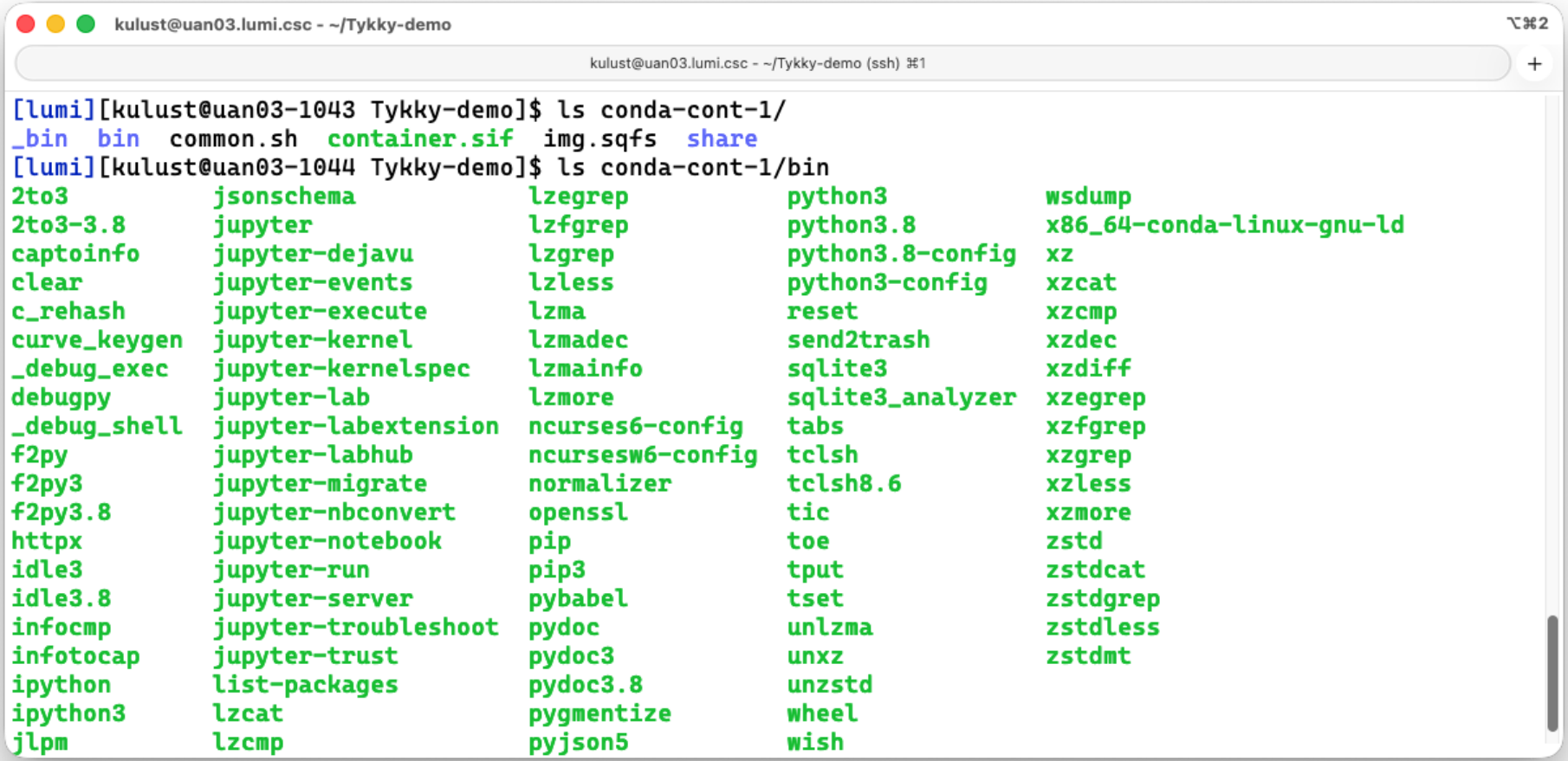
lumi-container-wrapper (3)



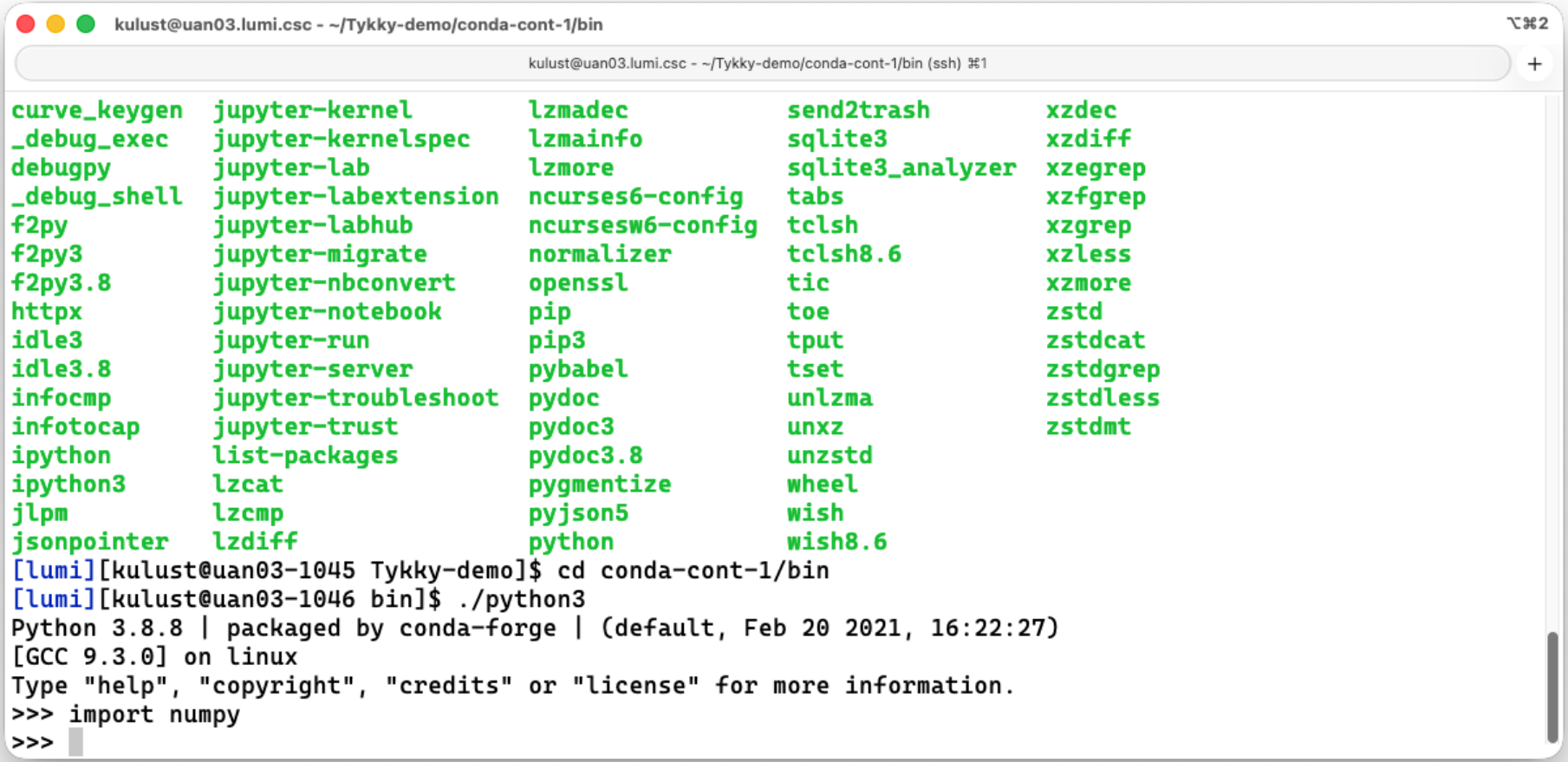
lumi-container-wrapper (4)



lumi-container-wrapper (5)



lumi-container-wrapper (6)



Environment enhancements (4): Non-AI containers

LUMI

- **lumi-vnc** (LUMI and CrayEnv software stacks)
  - Provides basic VNC virtual desktop for interacting with graphical interfaces via a web browser or VNC client
  - Open OnDemand a better alternative for many
- **ccpe**: Containerised Cray Programming Environment
  - For advanced users only
  - User-installable as often customisations are needed
  - Experiment with newer versions of the Cray PE
  - Functionality may be limited due to ROCm driver compatibility and compatibility of MPI with the network software stack

Environment enhancements (5): AI containers

LUMI

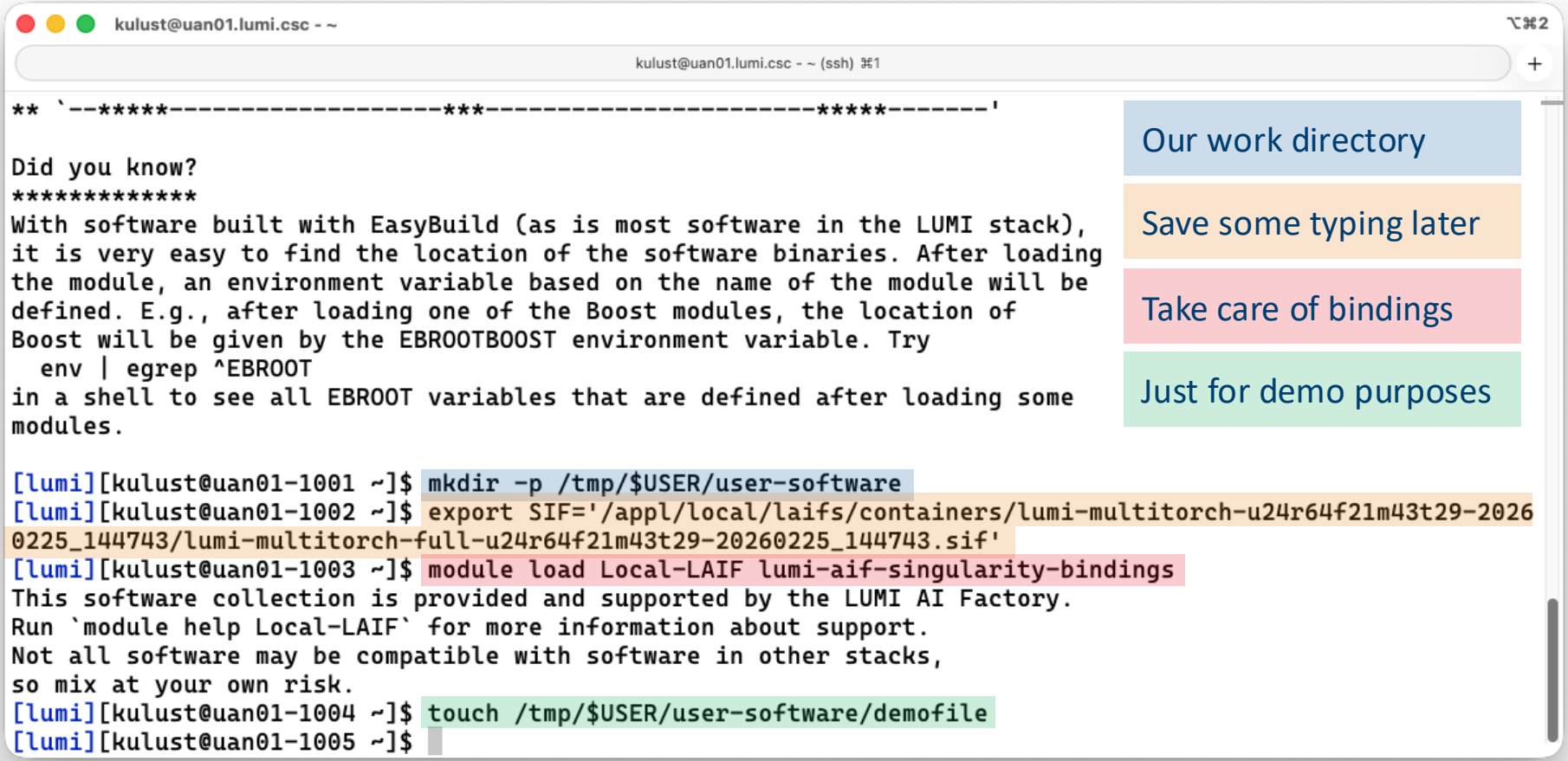
- AI containers built by the LUMI AI Factory
  - Currently PyTorch-based
  - Hierarchy of containers:
    - `lumi-multitorch-rocm-*`: Base container based on Ubuntu with ROCm
    - `lumi-multitorch-libfabric-*`: Adding libfabric on top of the previous one
    - `lumi-multitorch-mpich-*`: Adding the MPICH MPI library with GPU support on top of the previous one
    - `lumi-multitorch-torch-*`: Adding PyTorch on top of the previous one
    - `lumi-multitorch-full-*`: Adding a selection of additional AI and ML libraries on top of the previous one
  - `/appl/local/laifs/containers`
- Older containers still available in `/appl/local/containers/sif-images`
  - And EasyBuild recipes for a module providing bindings and some wrapper scripts
  - Likely removed after the January 2027 system update as they won't work anymore

Extending containers 1: Virtual environment

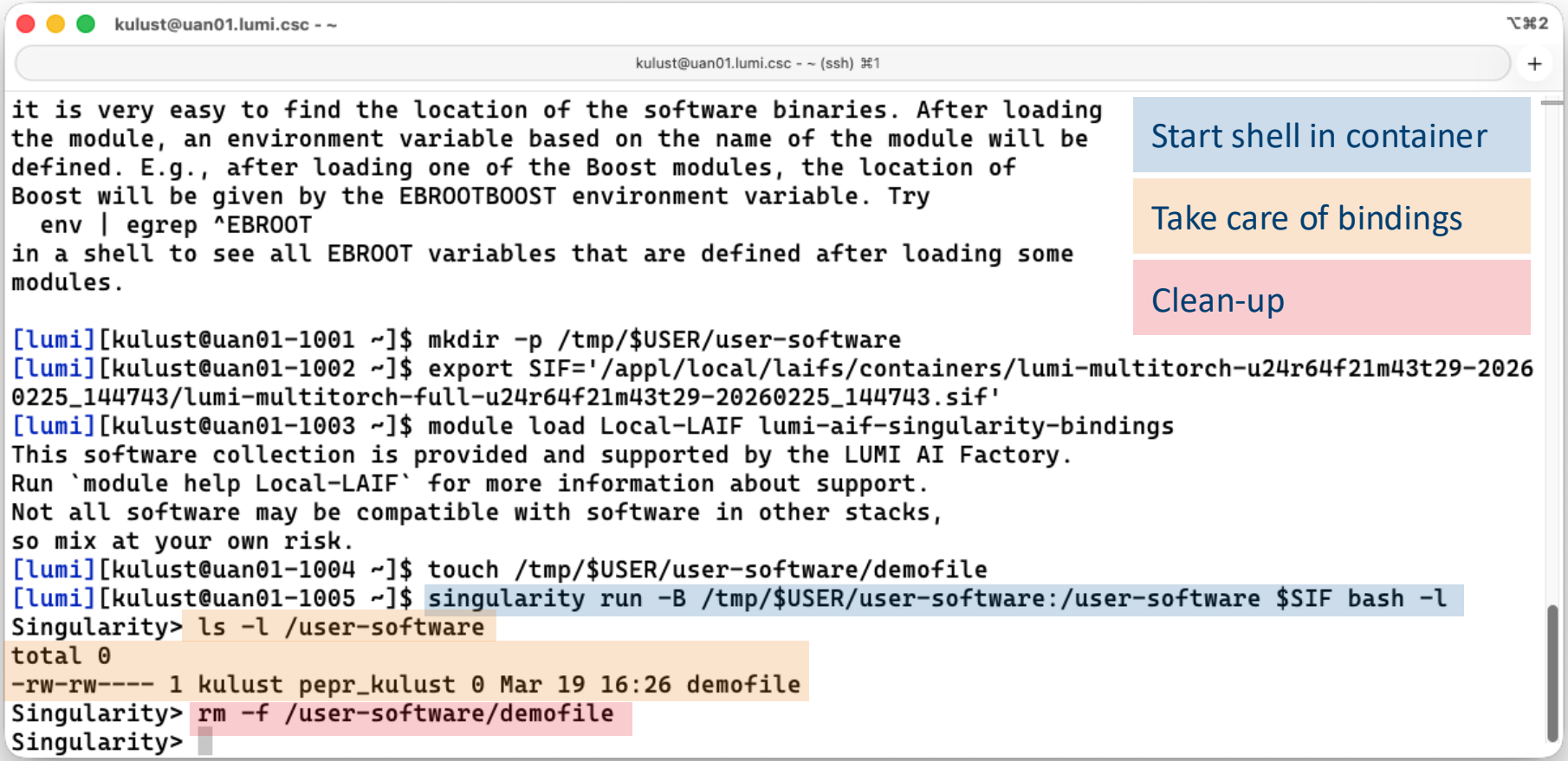
LUMI

- This strategy does not require rebuilding the container.
- Python has a mechanism to extend a read-only installation: virtual environments.
- Idea:
  - Create a bind mount for the virtual environment
    - Want a fixed path to be able to share with other projects
  - Create the virtual environment and install packages from within the container
    - Problem: The many-small-files problem returns…
  - Make a SquashFS file from that virtual environment
  - For further use, bind mount that SquashFS file instead of the directory
    - So the many-small-files problem is solved again

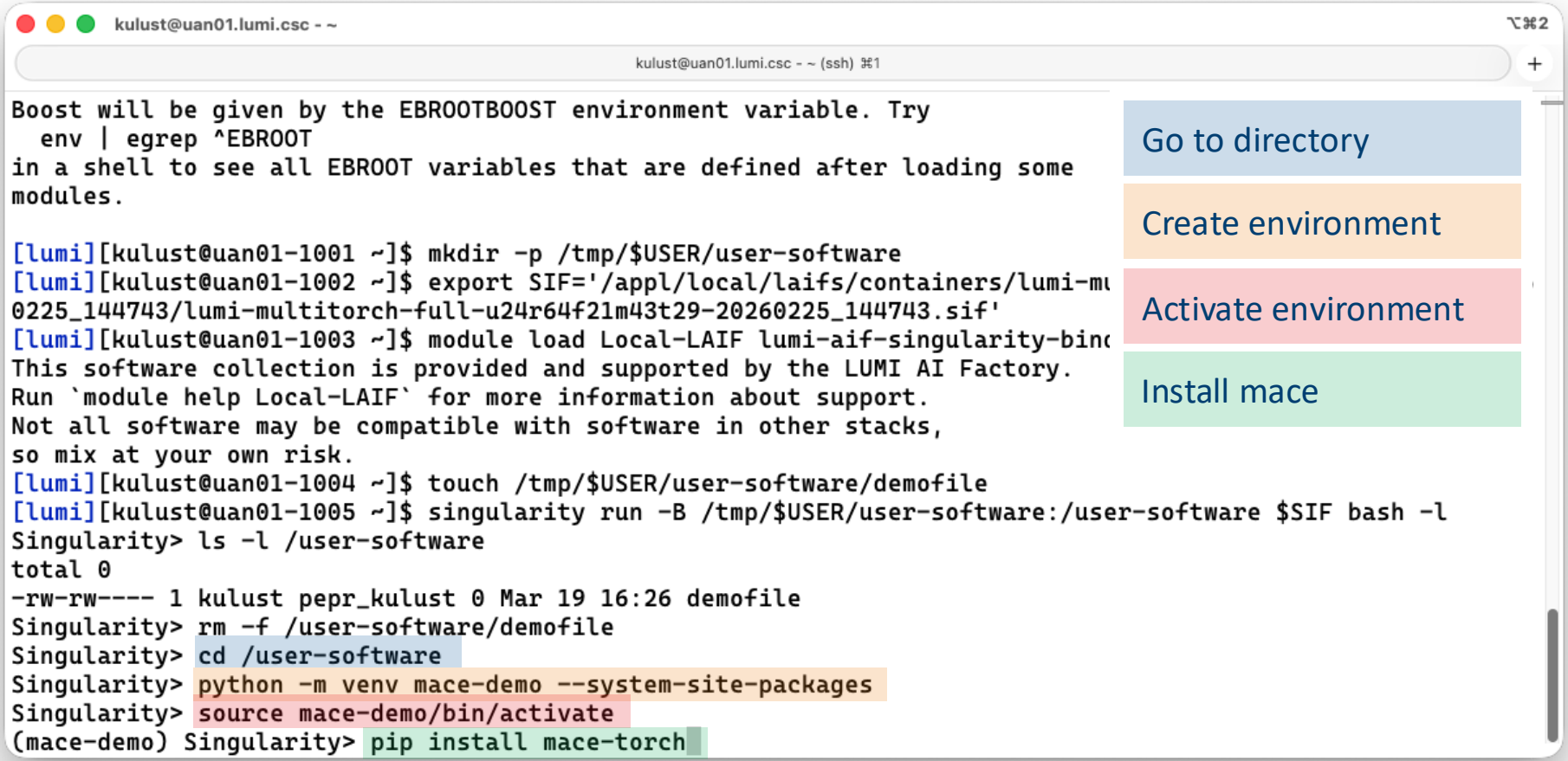
Step 1: Preparation



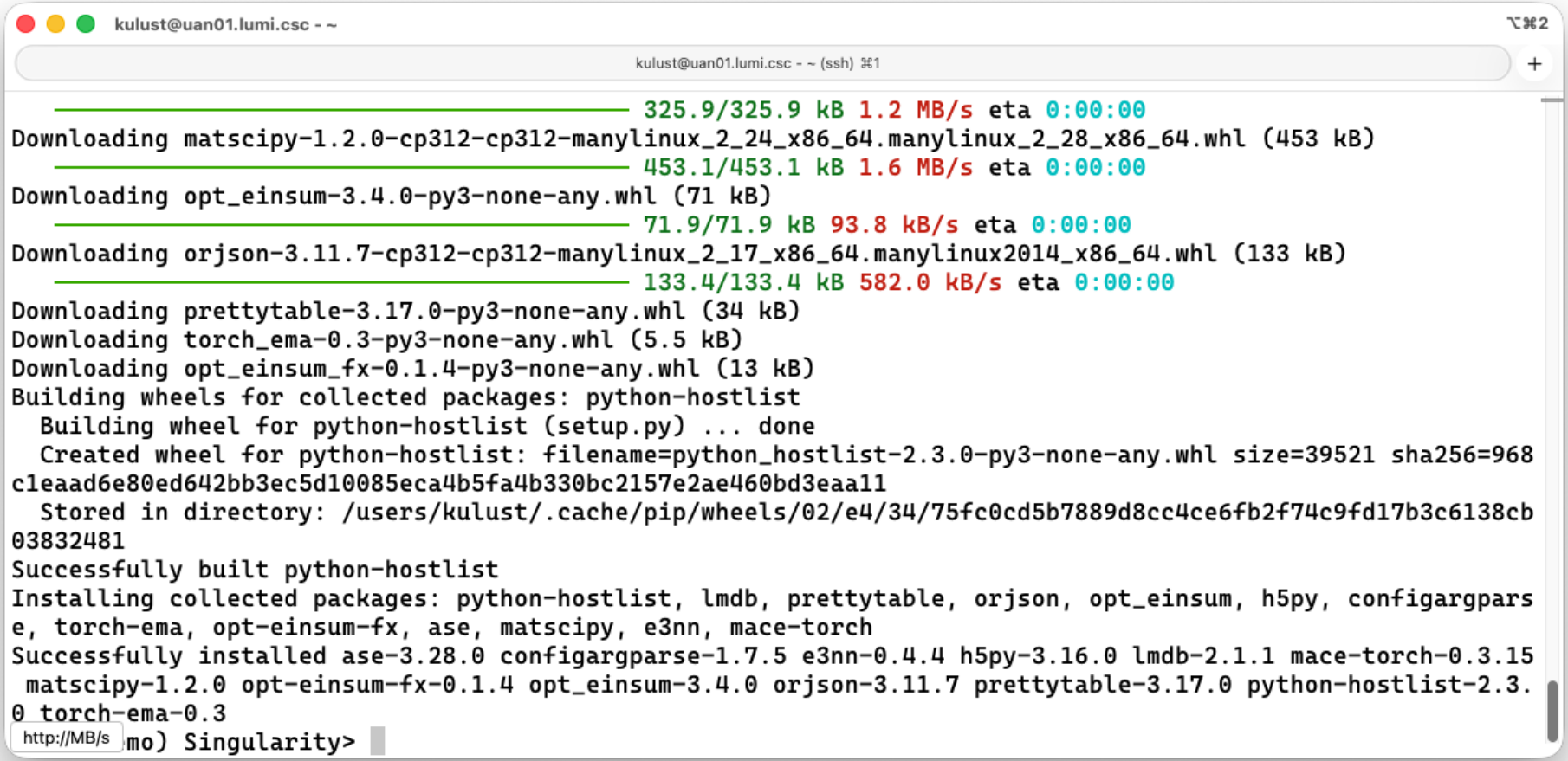
Step 2: Enter the container



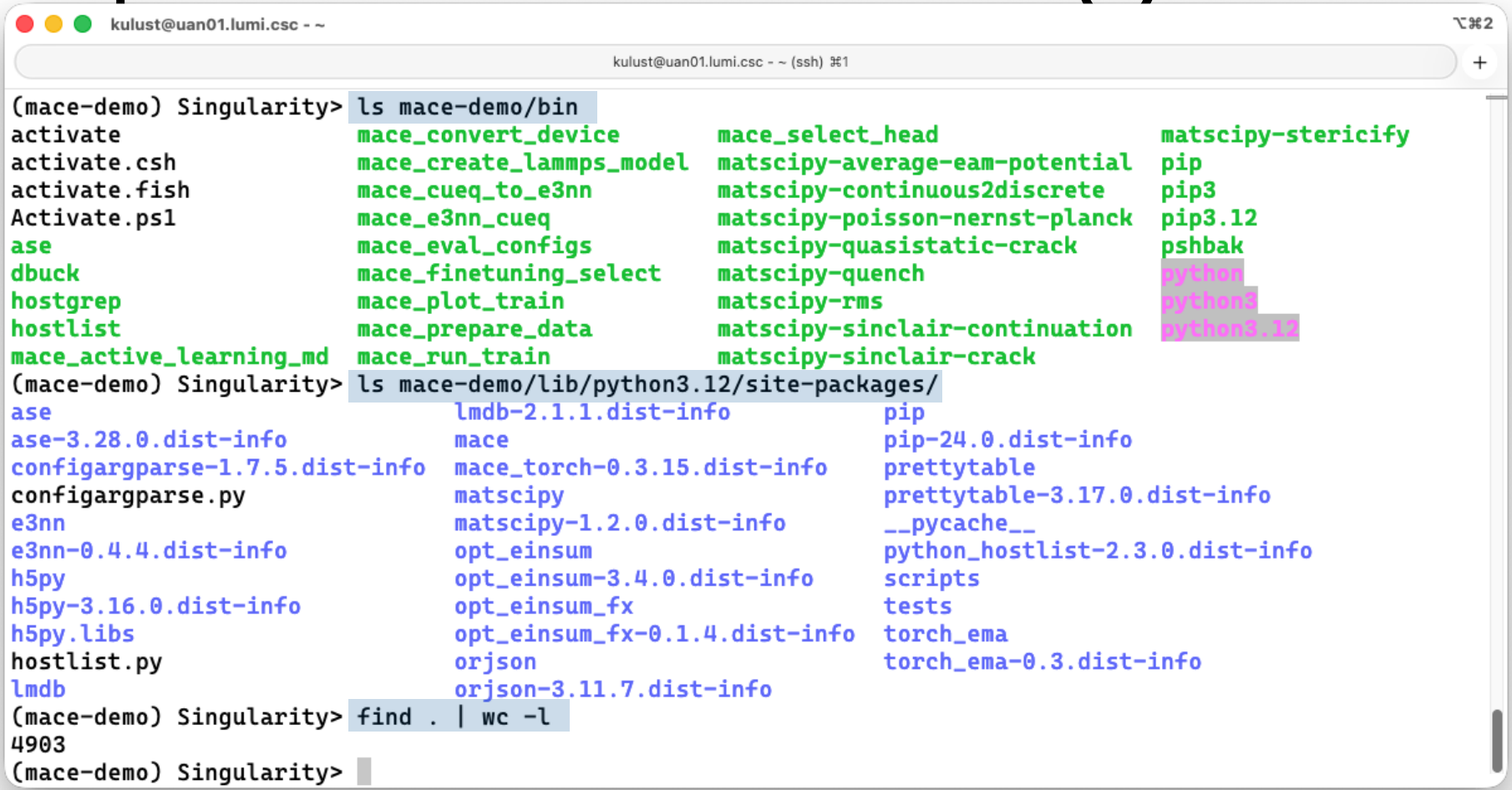
Step 3: Create the venv and install (1)



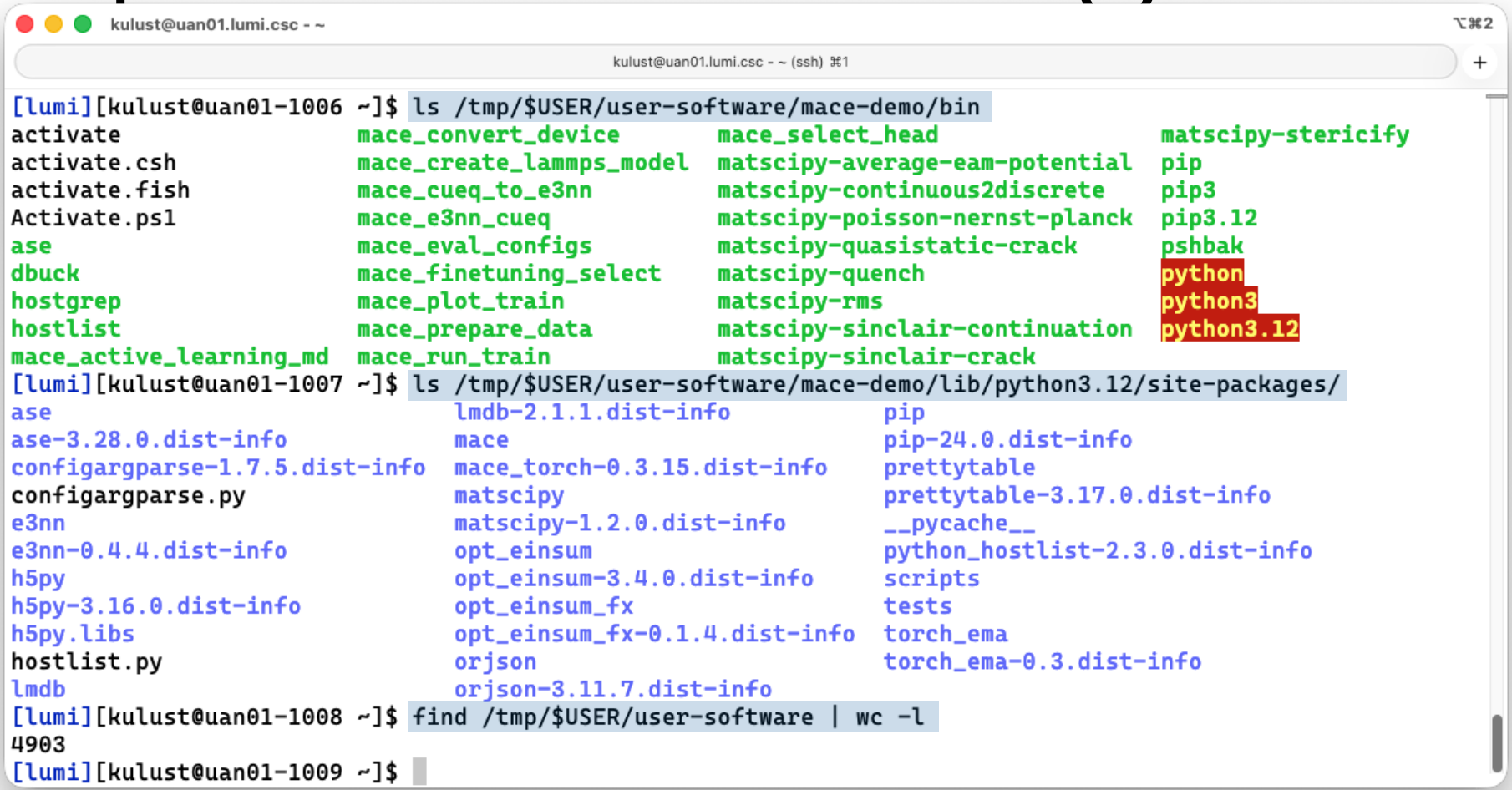
Step 3: Create the venv and install (2)



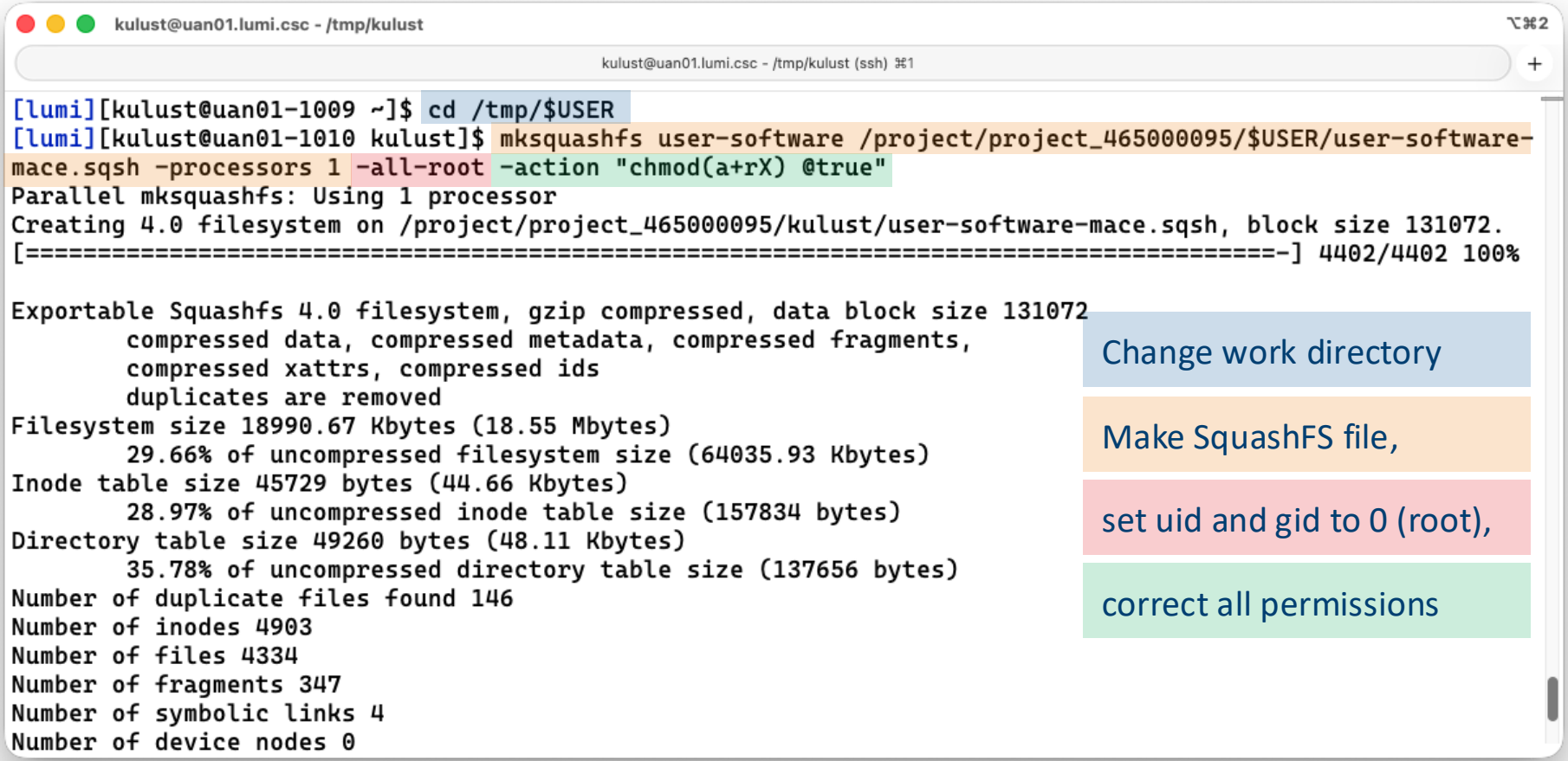
Step 3: Create the venv and install (3)



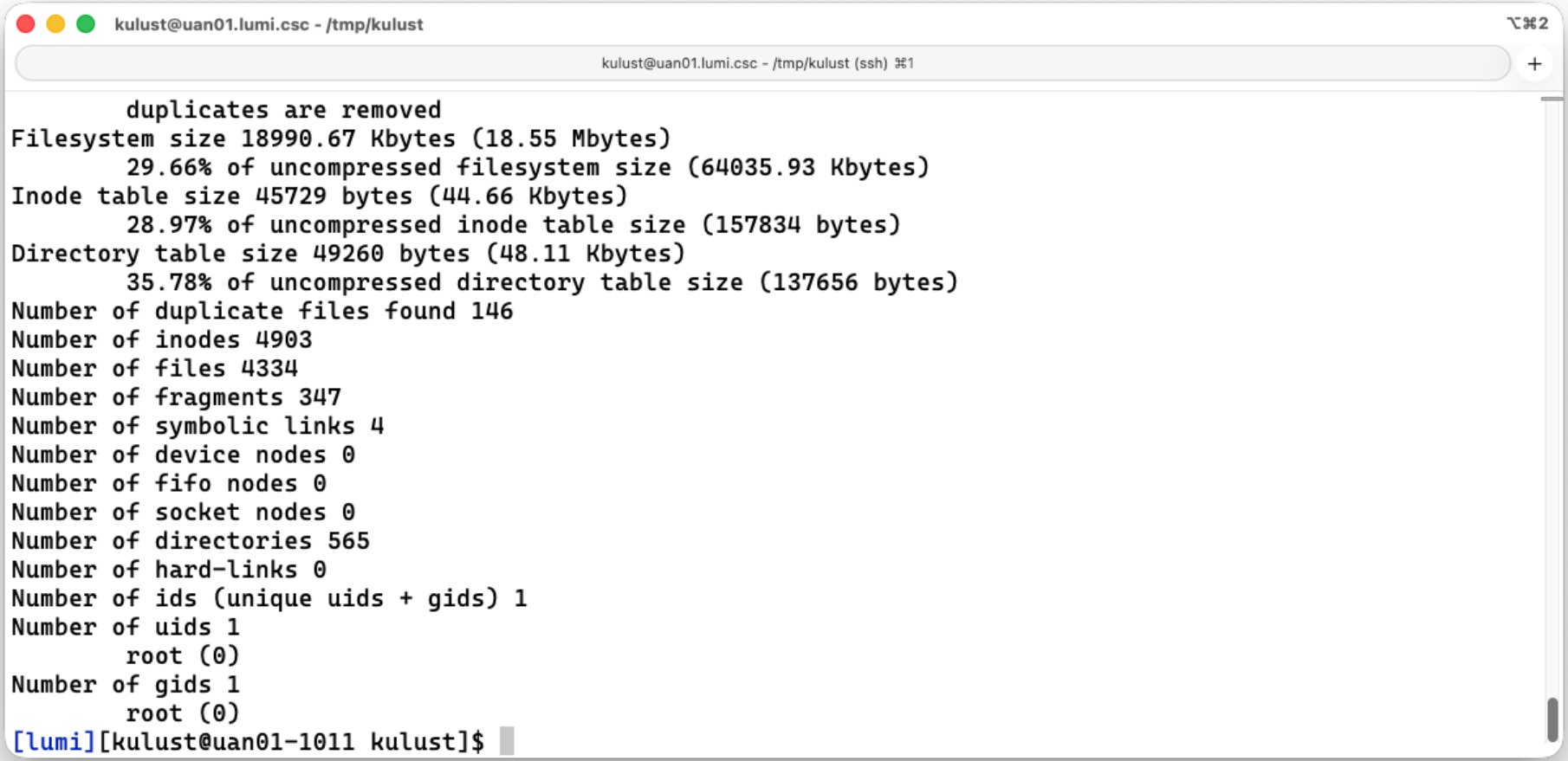
Step 3: Create the venv and install (4)



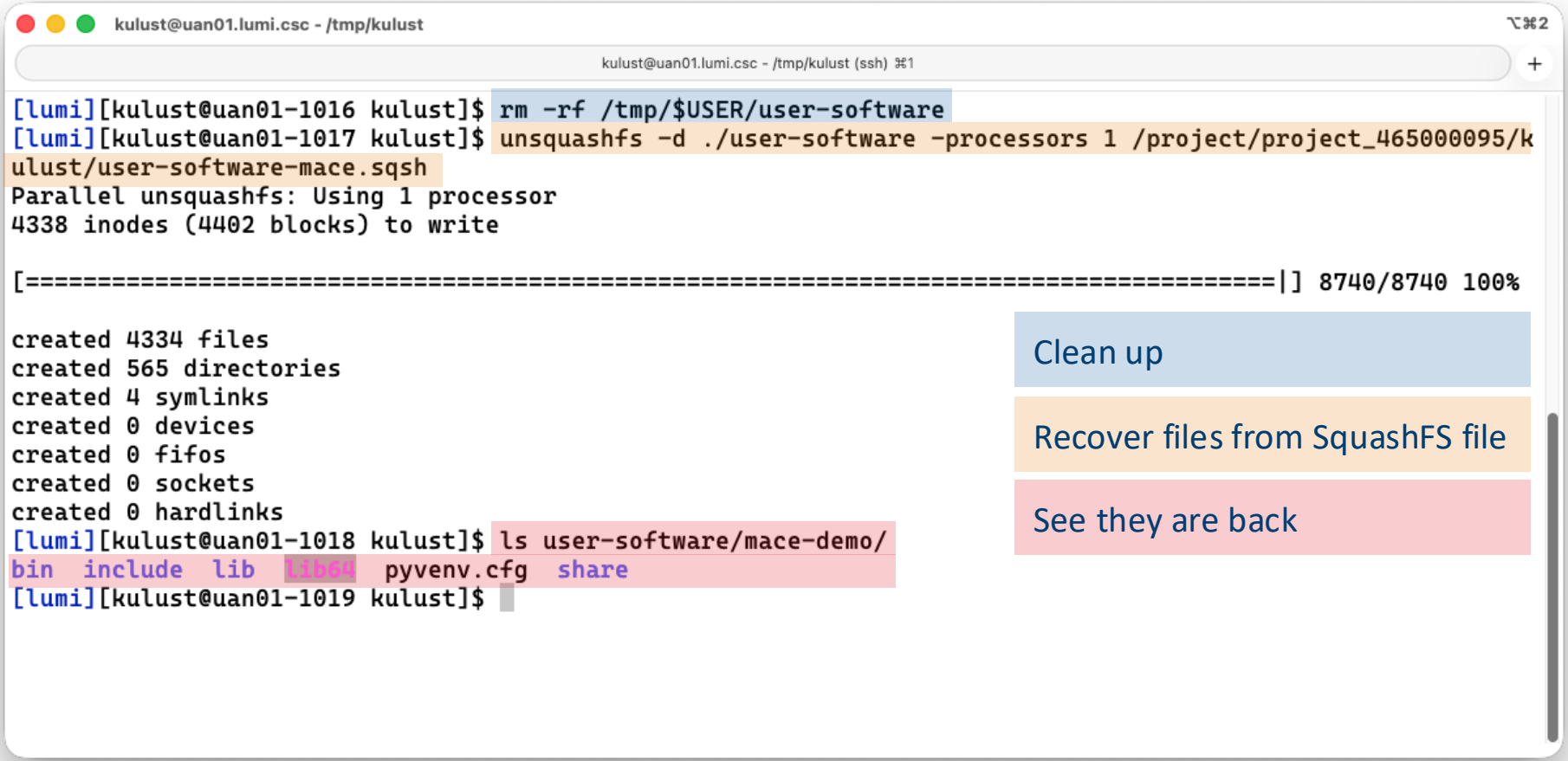
Step 4: Create the SquashFS file (1)



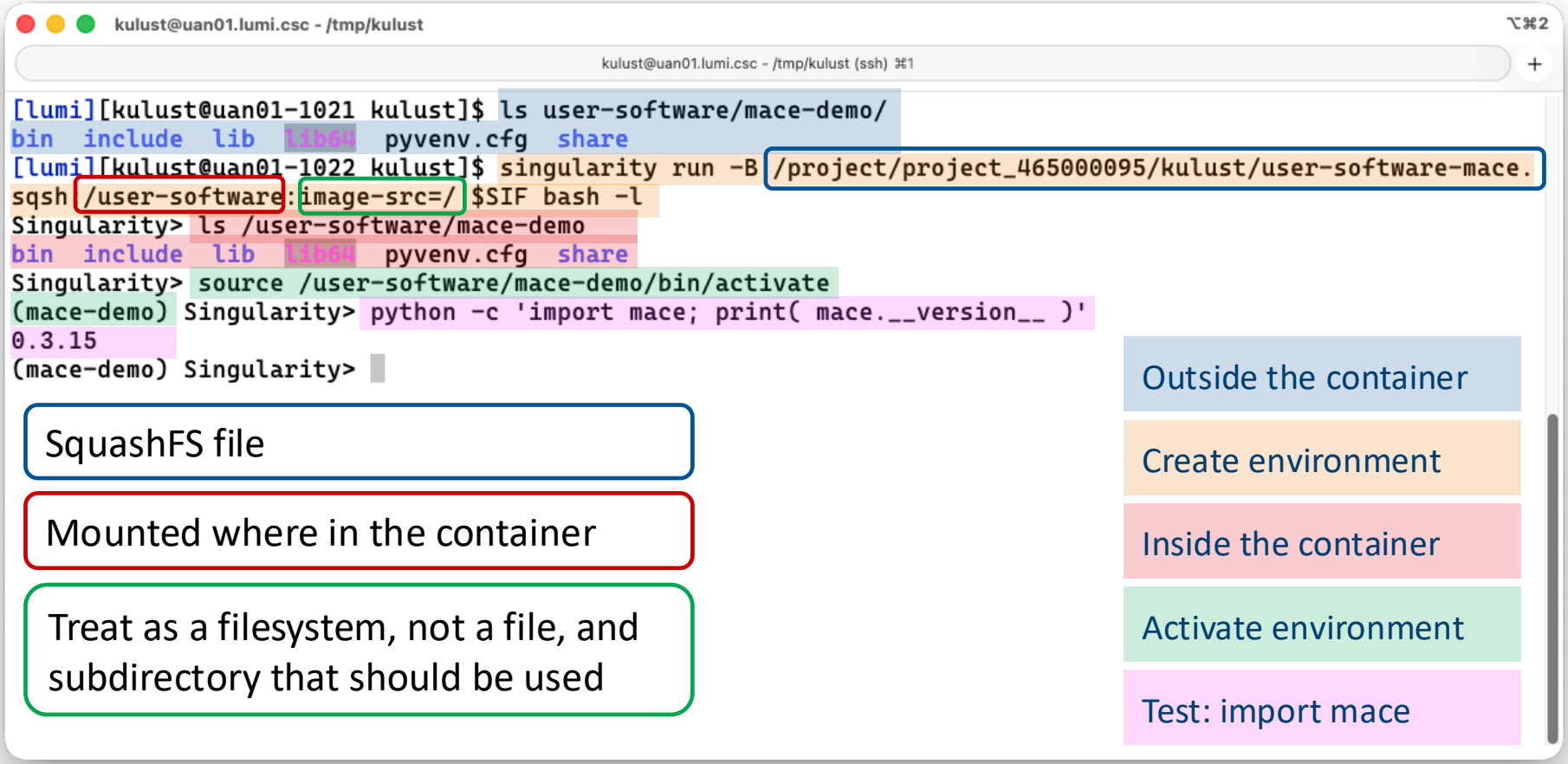
Step 4: Create the SquashFS file (2)



Step 4: Create the SquashFS file (3)



Step 5: Use the software



Extending containers 2: Unprivleged PRoot build

LUMI

- With this strategy, we build a new container from an existing one.
- What can be done on LUMI is a bit limited as only the PRoot process can be used.
- But we can:
  - Build from an existing container on the filesystem or from dockerhub, but not all bootstrap options work
  - Copy files and directories from the host so that we don't need to bind them
  - Add preset environment variables
  - Install additional OS packages, compile software, install additional Python packages, …
  - Change the runscript
- Steered by a definition file
  - See the examples on the next slides
  - Header, `%files`, `%environment`, `%post`, `%runscript`, …

Example 1: Python package mace-torch

mace.def

```
Bootstrap: localimage
From: {{ sif }}
```

Header

Template, replace with concrete filename at build time

```
%environment
    export NCCL_SOCKET_IFNAME='hsn0,hsn1,hsn2,hsn3'
```

Initialise this variable when starting the container

```
%post
    pip install mace-torch
```

Commands to execute. The virtual environment `/opt/venv` is already initialised when running these commands.

Example 1: Python package mace-torch (2)

LUMI

- Build new container without using the bindings module:

```
srun -pstandard -N1 -t2:00:00 --pty \
    singularity build \
        --build-arg sif=$SIF \
        lumi-multitorch-mace-u24r64f21m43t29-20260423.sif mace.def
```

  - `--build-arg sif=$SIF` : Set the value for the template `{{ sif }}`
  - Memory-intensive and long build time
- To do a build with your regular filesystems mounted: See the bonus examples in the notes.
- Demo:

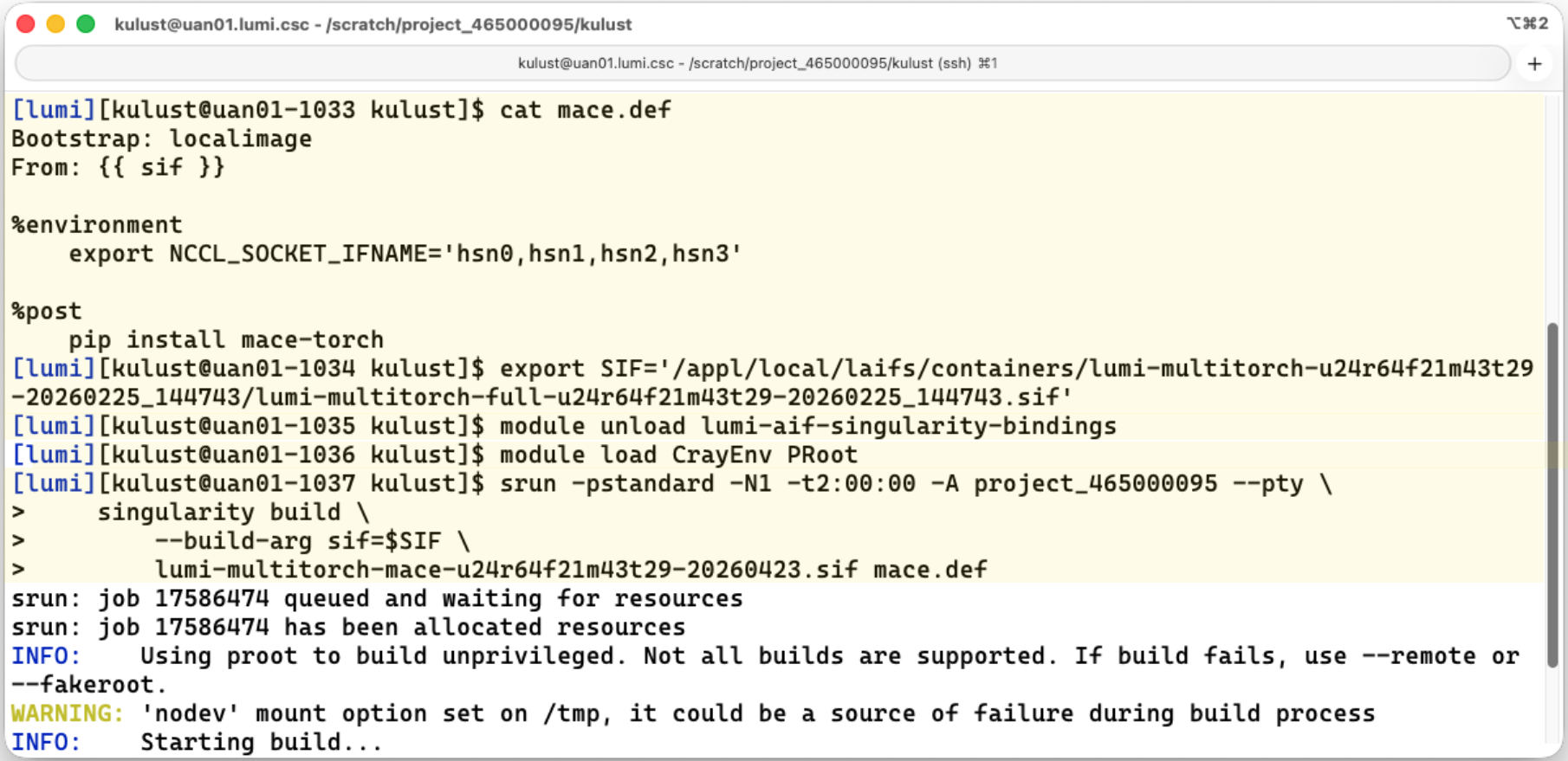
Big container, so using a compute node

Define the template in the definition file

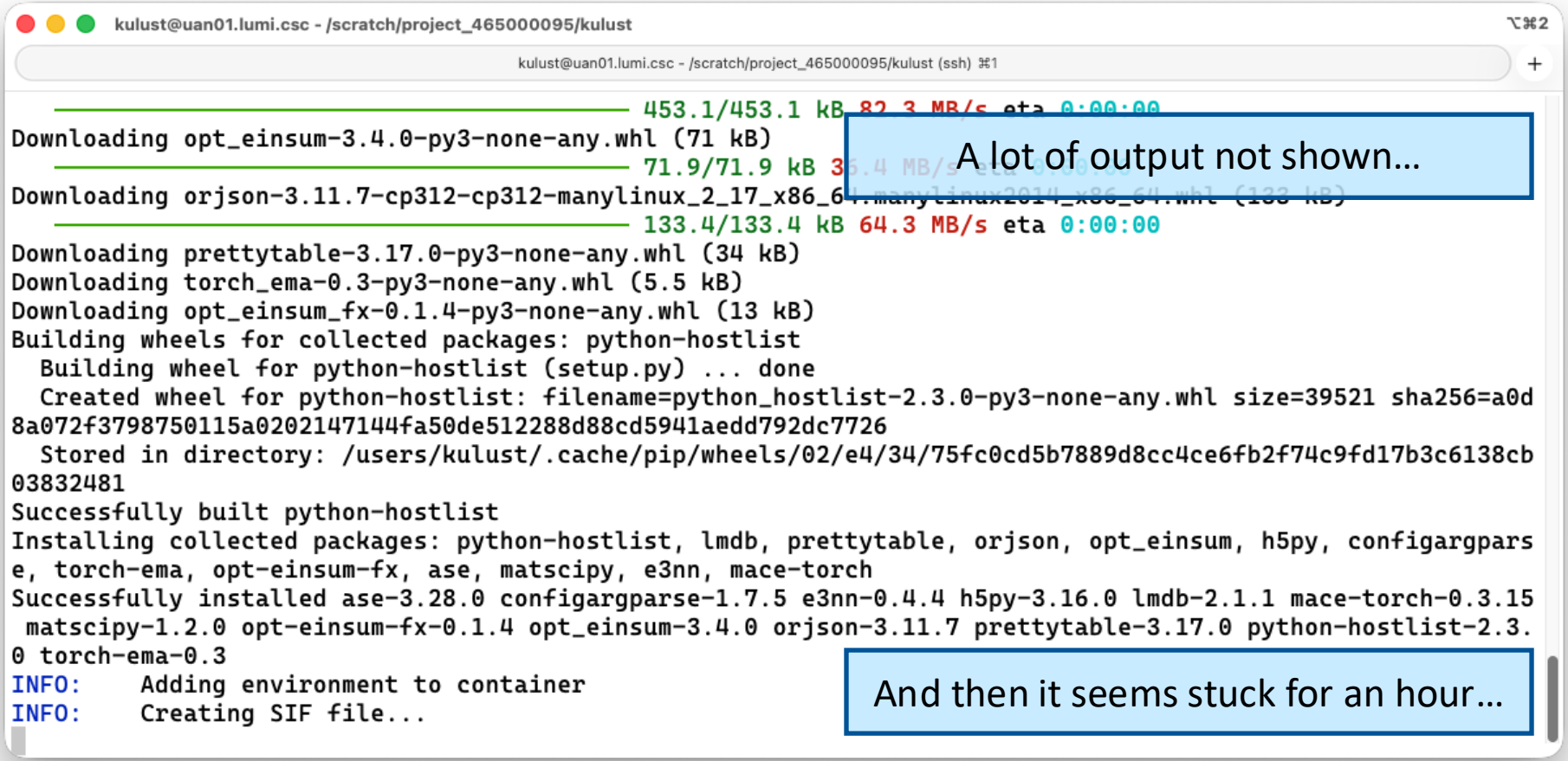
Name for the new container

Definition file

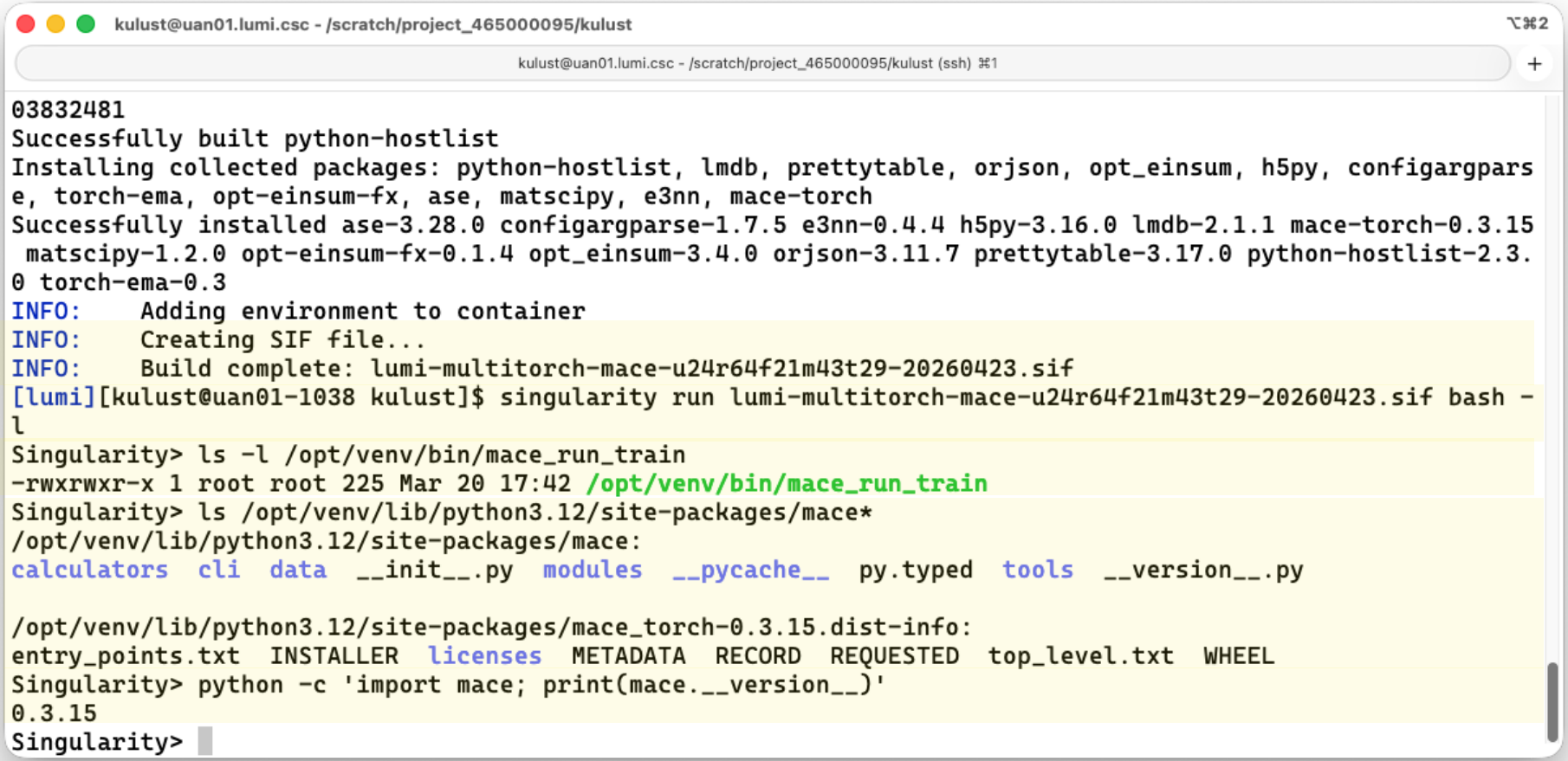
Example 1: Python package mace-torch (3)



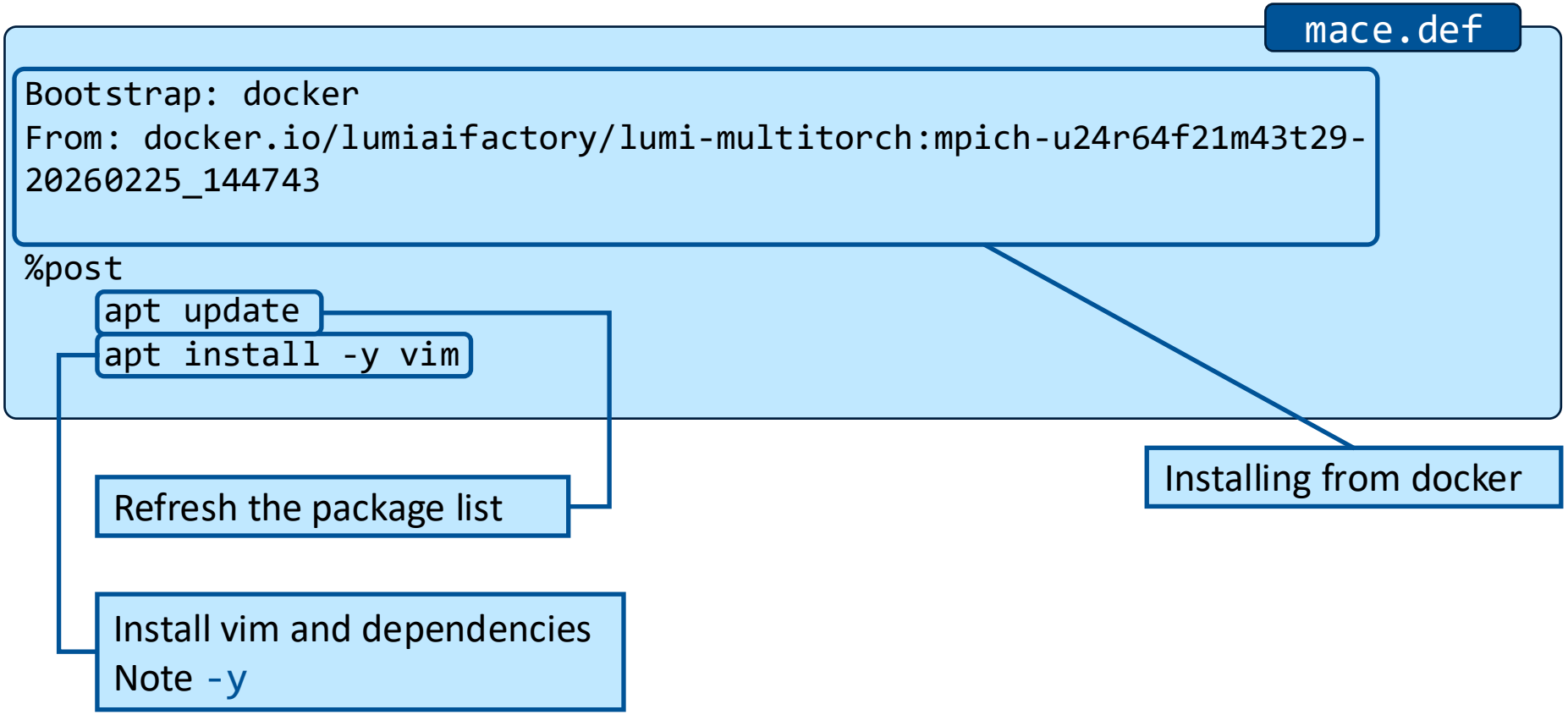
Example 1: Python package mace-torch (4)



Example 1: Python package mace-torch (5)



Example 2: Install Ubuntu application (1)



Example 2: Install Ubuntu application (2)

LUMI

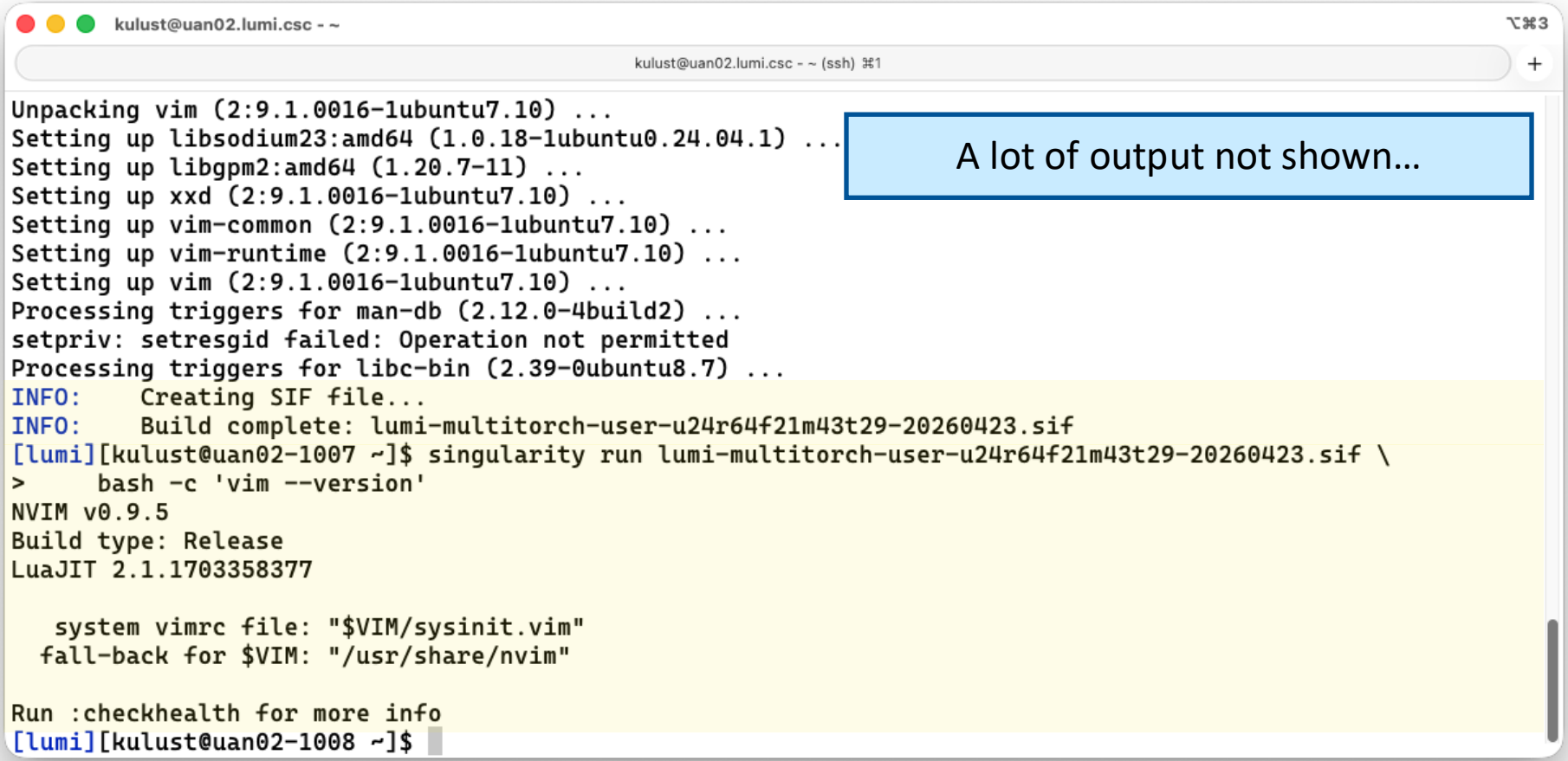
kulust@uan02.lumi.csc - ~ ⌥⌘3

kulust@uan02.lumi.csc - ~ (ssh) ⌘1

```
[lumi][kulust@uan02-1004 ~]$ cat userexts.def
Bootstrap: docker
From: docker.io/lumiaifactory/lumi-multitorch:mpich-u24r64f21m43t29-20260225_144743

%post
    apt update
    apt install -y vim
[lumi][kulust@uan02-1005 ~]$ module load LUMI PRoot
[lumi][kulust@uan02-1006 ~]$ singularity build \
>     lumi-multitorch-user-u24r64f21m43t29-20260423.sif userexts.def
INFO:    Using proot to build unprivileged. Not all builds are supported. If build fails, use --remote or
--fakeroot.
INFO:    Starting build...
INFO:    Fetching OCI image...
51.9MiB / 51.9MiB [============================================================================] 100 % 57.8 MiB/s 0s
51.3MiB / 51.3MiB [============================================================================] 100 % 57.8 MiB/s 0s
5.4GiB / 5.4GiB [==============================================================================] 100 % 57.8 MiB/s 0s
28.3MiB / 28.3MiB [============================================================================] 100 % 57.8 MiB/s 0s
3.8MiB / 3.8MiB [==============================================================================] 100 % 57.8 MiB/s 0s
55.7MiB / 55.7MiB [============================================================================] 100 % 57.8 MiB/s 0s
INFO:    Extracting OCI image...
INFO:    Inserting Singularity configuration...
INFO:    Running post scriptlet
```

Example 2: Install Ubuntu application (3)



Bonus examples in the notes

LUMI

- What if we want bind mounts during the `%post` processing?
  - Matching file or directory needed already
  - Trick: Abuse `%files` to copy an empty file or directory
- Can we call slurm commands in the container?
  - Yes, but it can be highly non-trivial, certainly if the OS in the container doesn't match the OS on LUMI
  - It may be dangerous to use a version of the client tools that does not match the version on LUMI
- What if installing a package does not work (example: FFmpeg)?
  - You'll have to be very creative
  - Look for other sources of precompiled binaries
  - Or install those dependencies that you can install, and proceed with manual compiles…
- Installing Python from scratch
  - Example with installing a package from sources
  - Relevant if you want to compile an application yourself also

Container limitations on LUMI

LUMI

- "Bring your own userland and run on a system-optimised kernel" and you'll be fine is nonsense.
- Containers use the host's operating system kernel which may be different from what the container expects. Containers also do not abstract hardware.
  - This is particularly true for ROCm as each version works only with certain driver versions
- Much of system-specific optimisation is done in userland:
  - Optimise the 99%, not the 1%
  - Optimisations for a specific CPU and GPU instruction set
  - A generic container may not offer sufficiently good support for the Slingshot 11 interconnect on LUMI and fall back to TCP sockets resulting in poor performance, or not work at all.
    - Solution: inject Cray MPICH, but only for containers with ABI compatibility with MPICH.
    - Distributed AI: Need to inject the proper RCCL plugin.
- Only limited support for building containers on LUMI due to security concerns.
  - Yet we can still do a lot.


LUMI
Questions?