ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ · ЛЕКЦИЯ 2

Linux и WSL для научных вычислений

От локального компьютера к серверу и HPC

Лю Шисян

Кафедра теплофизики (Э6)
МГТУ им. Н.Э. Баумана
Москва / 2026

Место этой лекции в курсе

Лекция 2 — Linux и вычислительная среда

Linux рассматривается как основная среда для работы с научными серверами и HPC-кластерами.

После лекции вы сможете:

- использовать WSL для работы с Linux в Windows;
- выполнять основные операции в Linux-терминале;
- запускать Python-программы из командной строки;
- подключаться к удалённому серверу по SSH;
- понимать базовый рабочий процесс HPC и Slurm.

План

TOC

1

Windows и WSL

Windows и Linux в научных вычислениях

Можно ли программировать в Windows?

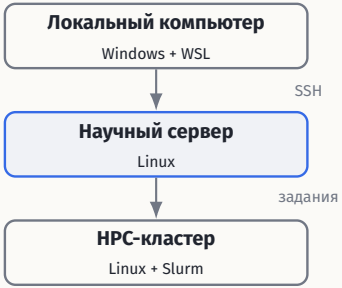
Да. Windows подходит для большинства учебных и многих научных задач:

- Python и Jupyter Notebook;
- VS Code и PyCharm;
- MATLAB, C/C++ и другие языки.

Почему тогда изучаем Linux?

Linux широко используется на:

- научных серверах;
- HPC-кластерах;
- системах пакетных расчётов.



Для нас

Команды и навыки Linux, изученные локально, затем используются почти без изменений на удалённых серверах и HPC-кластерах.

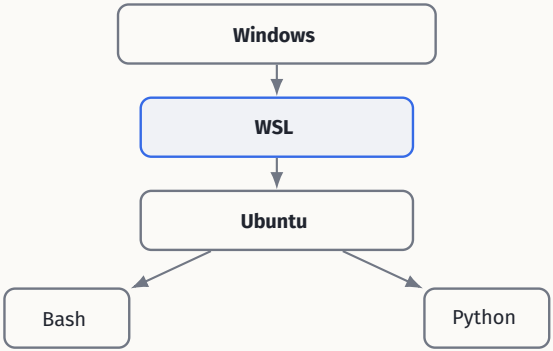
Что такое WSL?

WSL — Windows Subsystem for Linux

WSL позволяет запускать Linux-среду непосредственно внутри Windows.

Это означает, что:

- Windows остаётся основной системой;
- внутри неё доступен Linux terminal;
- можно установить Ubuntu;
- доступны Bash, Python, компиляторы и другие Linux-инструменты;
- те же команды затем используются на удалённых Linux-серверах.



Windows + WSL = Windows и Linux на одном компьютере

Установка WSL в Windows

Установка

1. Откройте PowerShell или Windows Terminal **от имени администратора**.

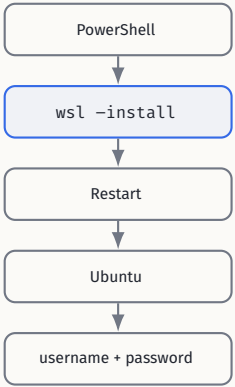
2. Выполните команду:

```
wsl --install
```

3. Перезагрузите Windows, если система попросит.

4. При первом запуске Ubuntu создайте:

- имя Linux-пользователя;
- пароль.



Как открыть WSL?

Вариант 1 — через Start

Найдите в меню Windows приложение **Ubuntu** и запустите его.

Вариант 2 — через терминал

В PowerShell или Windows Terminal выполните:

```
wsl
```

Проверить установленные дистрибутивы:

```
wsl --list --verbose
```

После входа в Linux

Выглядеть так:

```
student@DESKTOP:~$
```

Здесь:

- `student` — имя пользователя;
- DESKTOP — имя компьютера;
- ~ — домашний каталог.

Первая команда:

```
pwd
```

После запуска WSL мы работаем уже в Linux-командной строке.

Где находятся файлы Windows?

Доступ к файлам Windows

В WSL диски Windows доступны через каталог `/mnt`.

Windows	WSL
C:\	/mnt/c/
D:\	/mnt/d/

Например:

```
cd /mnt/c/Users
ls
```

Это позволяет работать с файлами Windows непосредственно из Linux.

Файлы Linux

Домашний каталог пользователя обозначается символом ~.

```
cd ~
pwd
```

Открыть файлы WSL в Windows

Текущий каталог WSL можно открыть в Проводнике Windows:

```
explorer.exe .
```

2

Основные операции Linux

Терминал: управление через команды

В графическом интерфейсе мы выбираем действие мышью, а в терминале явно записываем его в виде команды.

Вопрос	Команда
Где я?	`pwd`
Что здесь?	`ls`
Куда перейти?	`cd`

Общий вид команды

```
команда [опции] [аргументы]
```

- команда — что выполнить;
- опции — как выполнить;
- аргументы — с чем выполнить действие.

Например:

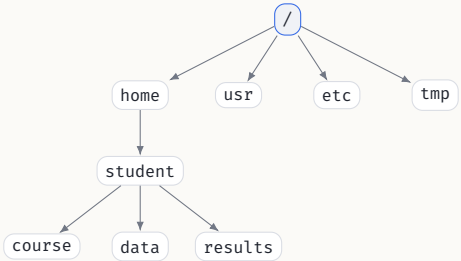
```
ls -l /home/student
```

Команды Linux лучше изучать, сразу выполняя их в WSL.

Файловая система Linux

Иерархическая структура

В Linux все файлы и каталоги образуют одно дерево, которое начинается с корневого каталога /.



Полезные обозначения

/	корневой каталог
~	домашний каталог
.	текущий каталог
..	родительский каталог

Пути

Абсолютный:

```
/home/student/course
```

Относительный:

```
course/data
```

Создание каталогов и файлов

Пример

Создадим простую структуру вычислительного проекта.

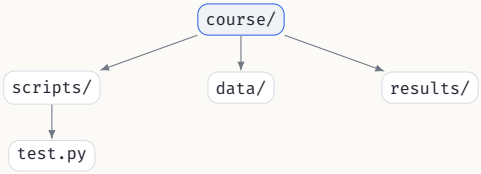
```
mkdir course
cd course

mkdir scripts
mkdir data
mkdir results

touch scripts/test.py
```

Проверим структуру:

```
ls
ls scripts
```



Две команды

`mkdir` — создаёт каталог.

`touch` — создаёт файл, если его ещё нет.

Навигация: pwd, ls, cd

Посмотреть, где мы находимся

```
pwd
```

Команда выводит путь к текущему каталогу.

Посмотреть содержимое каталога

```
ls
ls -l
ls -a
```

- `-l` — подробная информация;
- `-a` — включая скрытые файлы.

Перейти в другой каталог

```
cd course
cd ..
cd ~
cd /home/student/course
```

Пример работы

```
pwd
ls
cd course
```

Копирование, перемещение и удаление

Копирование: cp

```
cp test.py test_backup.py
```

Создаётся копия файла.

Перемещение и Переименование: mv

```
mv test.py scripts/
```

Файл перемещается в другой каталог.

```
mv old.py new.py
```

Команда mv также используется для переименования.

Удаление файла: rm

```
rm file.txt
```

Удаление каталога

```
rm -r folder
```

Опция -r удаляет каталог вместе с его содержимым.

Осторожно

Команда rm обычно удаляет данные без привычной корзины.

Просмотр и поиск в файлах

Просмотр содержимого

```
cat result.txt
less result.txt
```

`cat` выводит файл целиком, а `less` удобен для просмотра больших файлов.

Начало и конец файла

```
head result.txt
tail result.txt
```

Поиск текста: `grep`

Найти строки, содержащие слово:

```
grep "error" simulation.log
```

Без учёта регистра:

```
grep -i "error" simulation.log
```

Для научных расчётов особенно важно уметь быстро просматривать и искать информацию в лог-файлах.

Мини-шпаргалка Linux

Действие	Команда	Пример
Текущий каталог	`pwd`	`pwd`
Список файлов	`ls`	`ls -l`
Перейти в каталог	`cd`	`cd results`
Создать каталог	`mkdir`	`mkdir data`
Создать файл	`touch`	`touch test.py`
Копировать	`cp`	`cp a.txt b.txt`
Переместить / переименовать	`mv`	`mv a.txt data/`
Удалить	`rm`	`rm a.txt`
Просмотреть файл	`cat, less`	`less output.log`
Найти текст	`grep`	`grep "error" log.txt`

Эти команды покрывают большинство базовых операций с файлами и каталогами в Linux.

3

Python в Linux

Установка Miniconda в WSL

Скачивание

Скачаем установщик Miniconda:

```
wget https://repo.anaconda.com/miniconda/
    Miniconda3-latest-Linux-x86_64.sh
```

Установка

Запустим установщик:

```
bash Miniconda3-latest-Linux-x86_64.sh
```

Следуйте инструкциям установщика.

После установки

Перезапустите терминал или выполните:

```
source ~/.bashrc
```

Проверим conda:

```
conda --version
```

Miniconda будет управлять Python и научными библиотеками внутри WSL.

Создание Python-окружения с conda

Создание окружения

Создадим отдельное окружение для курса:

```
conda create -n numcalc python=3.12
```

Активируем его:

```
conda activate numcalc
```

Выход из окружения

```
conda deactivate
```

Проверка

После активации:

```
which python
python --version
```

Путь будет связан с Miniconda, например:

```
~/miniconda3/envs/numcalc/bin/python
```

Каждое conda-окружение имеет собственный Python и собственные библиотеки.

Установка научных библиотек

Установка

Сначала активируем окружение:

```
conda activate numcalc
```

Затем установим библиотеки:

```
conda install numpy scipy matplotlib
```

Проверка

```
python
```

В Python:

```
import numpy as np
print(np.__version__)
```

Посмотреть окружения

```
conda env list
```

Активированное окружение определяет, какие библиотеки доступны программе.

Запускаем Python-расчёт в WSL

Файл `scripts/calculation.py`

```
import numpy as np
x = np.linspace(0, 1, 5)
y = x ** 2
print(y)
```

Структура проекта

```
course/
|-- scripts/
|   '-- calculation.py
|-- data/
'-- results/
```

Запуск

Перейдём в каталог проекта:

```
cd ~/course
```

Активируем окружение:

```
conda activate numcalc
```

Запустим программу:

```
python scripts/calculation.py
```

Сохраняем результаты расчёта

При работе на сервере расчёт часто выполняется без постоянного наблюдения пользователя, поэтому вывод программы удобно сохранять в файл.

Перенаправление вывода

Сохранить вывод в файл:

```
python scripts/calculation.py \
  > results/output.txt
```

Добавить вывод в конец файла:

```
python scripts/calculation.py \
  >> results/output.txt
```

Просмотр результата

Показать содержимое файла:

```
cat results/output.txt
```

Для длинного файла:

```
less results/output.txt
```

Последние строки:

```
tail results/output.txt
```

VS Code + WSL + conda

Рабочий процесс

1. VS Code установлен в Windows;
2. установлено расширение **WSL**;
3. проект хранится внутри WSL;
4. активируется conda-окружение;
5. Python запускается внутри Linux.

Откроем проект из WSL:

```
cd ~/course
conda activate numcalc
code .
```

Редактор работает в Windows, а проект, conda и Python — внутри WSL.

4

Практический пример: NEP-kappa

Практический пример: NEP-kappa

Задача

Рассчитаем решёточную теплопроводность объёмного кремния с помощью готового научного Python-пакета **NEP-kappa**.

Что делает NEP-kappa?

1. релаксация структуры;
2. расчёт силовых констант;
3. расчёт теплопроводности;
4. визуализация результатов.

Что мы используем?

- WSL / Linux;
- Git;
- conda;
- Python;
- NEP-потенциал;
- phonopy / phono3py.

исходная структура → силы → силовые константы → теплопроводность

Шаг 1: получаем и устанавливаем NEP-kappa

Скачать проект

Перейдём в домашний каталог и клонируем репозиторий:

```
cd ~
git clone https://github.com/lyushisyan/
    NEP-kappa.git
cd NEP-kappa
```

Создать окружение

```
conda create -n nepkappa python=3.12
conda activate nepkappa
```

Установить пакет

Из корня репозитория:

```
python -m pip install -e .
```

Проверим установку:

```
nepkappa --help
```

Что произошло?

Команда `pip install -e .` установила пакет вместе с его Python-зависимостями.

Шаг 2: задаём параметры расчёта

```
structure:
  poscar: examples/POSCAR_bulk

calculator:
  name: nep
  nep_model: potentials/Si_Bulk_Fan.txt

relaxation:
  enabled: true

force-constant:
  dim-fc2: [3, 3, 3]
  dim-fc3: [3, 3, 3]
  use_hiphive: false

kappa:
  mesh: [21, 21, 21]
  temps: [100, 1000, 50]
  method: rta

output:
  result_dir: results/1-bulk-nep-rta
```

Основные параметры

- POSCAR_bulk — структура Si;
- Si_Bulk_Fan.txt — NEP-потенциал;
- $3 \times 3 \times 3$ — суперячейки;
- $21 \times 21 \times 21$ — q-сетка;
- rta — метод расчёта теплопроводности.

Результаты

Все выходные файлы будут записаны в

```
results/1-bulk-nep-rta
```

Шаг 3: проверяем и запускаем расчёт

Проверка конфигурации

Перед расчётом:

```
nepkappa info examples/1-bulk-nep-rta.yaml
```

Это позволяет проверить прочитанные параметры до запуска вычислений.

Полный расчёт

Запускаем весь workflow:

```
nepkappa run examples/1-bulk-nep-rta.yaml
```

Одна команда запускает последовательность научных вычислений.

Что происходит внутри расчёта?

1. Релаксация структуры

```
nepkappa relax examples/1-bulk-nep-rta.yaml
```

2. Силовые константы

```
nepkappa fc2fc3 examples/1-bulk-nep-rta.yaml
```

3. Теплопроводность

```
nepkappa kappa examples/1-bulk-nep-rta.yaml
```

4. Визуализация

```
nepkappa plot examples/1-bulk-nep-rta.yaml
```

5

Сервер и Slurm

От локального WSL к удалённому серверу

Локально

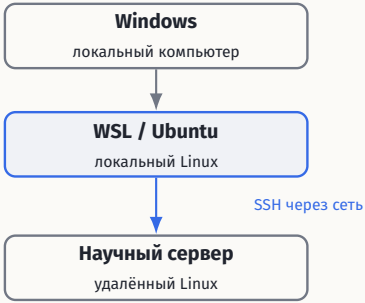
В WSL мы уже умеем:

- работать с Linux-командами;
- использовать файловую систему;
- создавать conda-окружения;
- запускать Python-программы;

На сервере

Среда очень похожа:

- Linux;
- terminal;
- Python / conda;
- те же команды и структура файлов.



WSL — локальная подготовка к работе на научном сервере.

Подключение к серверу через SSH

SSH — Secure Shell

SSH позволяет открыть терминал удалённого Linux-компьютера через сеть.

Общий формат:

```
ssh username@server
```

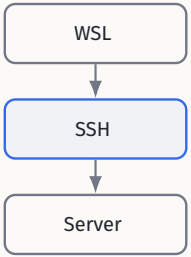
После подключения проверим:

```
whoami
hostname
pwd
ls
```

Что изменилось?

До SSH команды выполнялись в WSL.
После SSH:

команды выполняются на сервере



Ноутбук становится интерфейсом для работы с удалённым компьютером.

Передача файлов между компьютерами

Команда `scp`

`scp` копирует файлы между локальным компьютером и удалённым сервером через SSH.

На сервер

Передадим входной файл:

```
scp input.txt user@server:~/Works/
```

WSL ⟶ Server

С сервера

Скачаем результаты:

```
scp user@server:~/Results/output.txt .
```

Server ⟶ WSL

Для каталога используется `-r`:

```
scp -r data/ user@server:~/project/
```

Почему на HPC-кластере нужен планировщик?

Один сервер

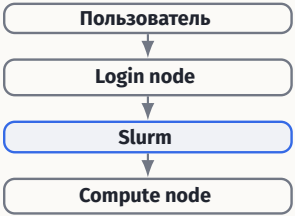
Пользователь может самостоятельно запустить программу:

$$\text{User} \longrightarrow \text{Program} \longrightarrow \text{CPU}$$

HPC-кластер

На кластере одновременно работают:

- много пользователей;
- много задач;
- сотни или тысячи CPU;
- GPU и другие ресурсы.



Slurm определяет:

- когда запустить задачу;
- на каком вычислительном узле;
- сколько CPU, памяти или GPU выделить.

На HPC мы обычно не запускаем тяжёлый расчёт напрямую — мы отправляем задачу в очередь.

Первая задача Slurm

Файл `job.sh`

```
#!/bin/bash

#SBATCH --job-name=nepkappa
#SBATCH --time=01:00:00
#SBATCH --cpus-per-task=1
#SBATCH --output=slurm-%j.out
#SBATCH --error=slurm-%j.err

source ~/miniconda3/etc/profile.d/conda.sh
conda activate nepkappa

cd ~/NEP-kappa
nepkappa run examples/1-bulk-nep-rta.yaml
```

Параметры Slurm

- `--job-name` — имя задачи;
- `--time` — максимальное время;
- `--cpus-per-task` — число CPU;
- `--output` — стандартный вывод;
- `--error` — сообщения об ошибках.

Номер задачи

`%j` заменяется идентификатором задачи:

```
slurm-12345.out
slurm-12345.err
```

Три основные команды Slurm

Команда	Назначение	Пример
`sbatch`	отправить задачу	`sbatch job.sh`
`squeue`	посмотреть очередь	`squeue -u $USER`
`scancel`	отменить задачу	`scancel 12345`

Отправка

```
sbatch job.sh
```

Типичный ответ:

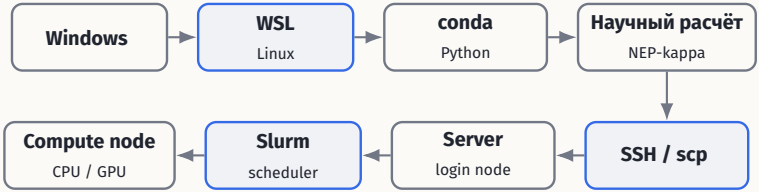
```
Submitted batch job 12345
```

Пример вывода `squeue`

```
JOBID PARTITION NAME     USER  ST TIME NODES
12345 compute   nepkappa sxliu PD 0:00 1
12346 compute   nepkappa sxliu R  2:31 1
```

- PD = **Pending** — задача ожидает запуска;
- R = **Running** — задача выполняется.

Что уже умеем после этой лекции?



Локальная вычислительная среда

- работать в Linux через WSL;
- управлять файлами из терминала;
- создавать conda-окружения;
- запускать Python и научные программы.

Удалённые вычисления

- подключаться к серверу через SSH;
- передавать файлы с помощью `scp`;
- создавать Slurm job scripts;
- запускать и контролировать задачи на HPC.

От локального WSL до HPC — единый рабочий процесс научных вычислений.

Следующая лекция

Теперь мы умеем

писать и запускать программы

Следующий вопрос:

Как строить численные алгоритмы?

Лекция 3 — Основные численные методы