

# The command line

*Controlling local and remote machines*



# Shell

- A shell exposes the operating system to a human user or other programs
- On most Linux systems, the default shell is **Bash** (**PowerShell** on Windows; **Zsh** on MacOs)
- Different shells have different syntaxes and characteristics



# Built-in commands

- Allow to operate a computer in a text-only way
- Built-in commands like `cd`, `ls`, `cp`, `mv`, and `rm` provide all features of a graphical file manager (e.g. Finder)
- Type `info <command>` for more information



# Built-in commands - Live Demo



# Coreutils

*The GNU core utilities*

- Collection of shell utilities that extend the built-ins
- Some useful commands are `sed`, `grep`, `awk`, `head`, `tail`, ...
- **Standard output** can be *piped* (`|`) to another command

```
1 ps aux | grep $USER
```

- Type `man <command>` to read the manual for a command



# Text editors

- Editors such as [Vim](#), [Emacs](#), or [nano](#) allow you to modify text files from the command line

```
1 vim test.txt
```

- Basic [Vim](#) usage:

Press `i` to activate *insert* mode. Type something. Press `Esc` to fall back into *normal* mode. Save and close the file using `:wq`

# Scripts

- You can bundle commands in a shell script, e.g.:

```
script.sh
```

```
1 hostname  
2 date
```

Running the script will execute the commands sequentially:

```
1 bash script.sh
```

```
1 levante4.lvt.dkrz.de  
2 Mon Mar 25 16:49:06 CET 2024
```

# Shebang #!

- By default a script is interpreted by the current shell
- You can specify an interpreter in the **shebang**

```
script.sh
```

```
1 #!/bin/bash
2 hostname
3 date
```

This allows you to use the script as an executable

```
1 chmod +x script.sh
2 ./script.sh
```



# Take-home messages

- The command line is a text-based interface for interacting with a computer
- Built-in commands and core utilities are available on (almost) all machines
- Sequences of commands can be stored and executed as shell scripts



# Hands-on session

1. Open the command line (*Terminal*<sup>1</sup>)
2. Create a shell script using an editor of your choice
3. Run the script and check the output
  1. More difficult for Windows users



# Remote machines



# Secure Shell (SSH)

- SSH allows you to connect to remote machines (e.g. Levante))
- You have to provide your user name and the full hostname:

```
1 ssh <YOUR_USERNAME>@levante.dkrz.de
```

- This will start a login shell on one of Levante's login nodes
- You have to enter your password on every login

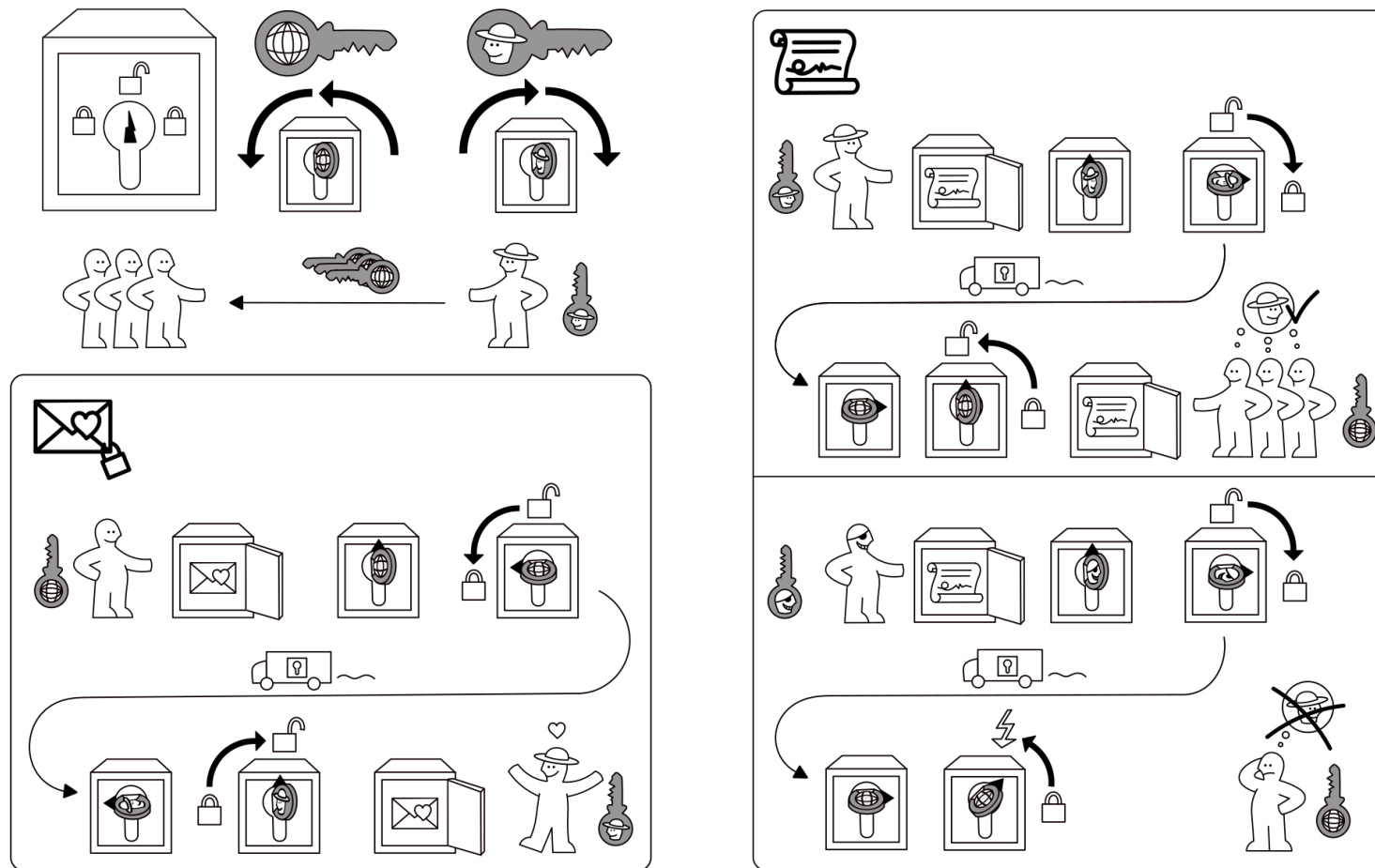


# Public keys

## PUBLIK KEY KRYPTO

idea-instructions.com/public-key/  
v1.2, CC by-nc-sa 4.0

**IDEA**



# Generating SSH keys

- Generate a private/public SSH key pair on your local system

```
1 ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519_levante
```

- Upload the public (`.pub`) key to your DKRZ profile ([how to](#))
- Use your private SSH key to authenticate when connecting to levante

```
1 ssh -i ~/.ssh/id_ed25519_levante <YOUR_USERNAME>@levante.dkrz.de
```



# Configuring SSH

- Create a config file for convenience

```
~/.ssh/config
1 Host levante
2     Hostname levante.dkrz.de
3     User <YOUR_USERNAME>
4     IdentityFile ~/.ssh/id_ed25519_levante
```

(call `man ssh_config` for more info)

- Connecting to Levante simplifies to

```
1 ssh levante
```

- The configuration is used by all tools using an SSH connection (*more than you think*)

# Hands-on session

1. Create an SSH key pair on your local machine
2. Upload the **public** key to `luv.dkrz.de`
3. Set up an entry for Levante in your SSH config file
4. Connect to Levante and inspect your home directory



# Remote file transfer

- There are various tools to transfer data between your local machine and a remote server (e.g. `scp` and `rsync`)
- `rsync` is a powerful option available on most machines

```
1 rsync local_file <YOUR_USERNAME>@levante.dkrz.de:
```



# Remote file transfer

- There are various tools to transfer data between your local machine and a remote server (e.g. `scp`, `rsync`, `uftp`)
- `rsync` is a powerful option available on most machines

```
1 rsync local_file levante:
```

# Hands-on session

1. Copy your shell script to levante
2. Execute the script on the login node



# Levante

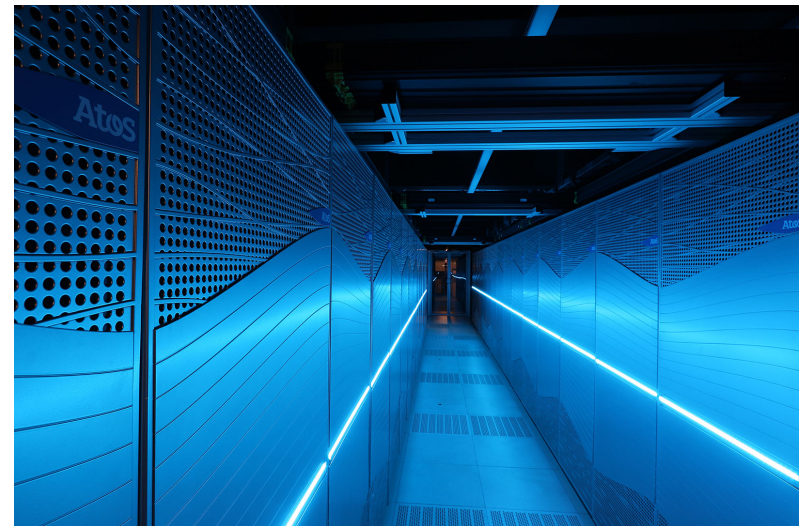


Levante is the current *supercomputer* at DKRZ (since 2022)

# Configuration

High-performance computers consist of various components

- Login (login nodes)
- Computing (CPU & GPU partitions)
- Storage (disks, tape, object store)
- Services (GitLab, JupyterHub, ...)



# Working on Levante

Different ways to work on a cluster, depending on the task

- Housekeeping/compiling (login nodes)
- Interactive sessions (compute nodes, `salloc`)
- Batch scripts (compute nodes, `sbatch`)
- External services (e.g. JupyterHub)



# Three parts of the file system

<code>/home</code>	<code>/work</code>	<code>/scratch</code>
keep scripts	store output	temporary stuff
small	big	big
SSD	HDD	HDD
backup	no backup	deleted after 2 weeks

Anything not saved will be lost — *Nintendo*



# Compute nodes

- Jobs that require more resources or run longer can be submitted to the job scheduler (**SLURM**)

```
1 sbatch --account=<PROJ_NUMBER> --partition=compute script.sh
```

- There are dedicated **partitions** for different use cases (e.g., **compute**, **shared**, **interactive**, ...)
- Try to use the smallest amount of resources (e.g. **shared** with **--mem=50G**)



# Hands-on session

1. Submit the script to the job scheduler
2. Check the output in the created log file



# Take-home messages

- High-performance computers like Levante are technically a cluster of numerous “normal” computers that share the same hard disks
- You can log into remote machines using a secure shell (`ssh`) and (mostly) use them like a local machine
- Computationally demanding or long-lasting jobs should be submitted to a job queue



# Shotgun buffet

I am just gonna throw a bunch of stuff at you.  
Take what you might find interesting.  
— *Scott Chacon*



# Integrated Development Environments (IDEs)

... are text editors on steroids

- Specific functionality for your programming language (tab completion, jump to definition, scope-aware renaming, debugger, auto documentation, ...)
- [VSCode](#), [PyCharm](#), [JupyterHub](#) (in a way), Vim + Emacs (with plugins), ...



# tmux

`tmux` allows you to create and attach to terminal sessions

- Create a new session

```
1 tmux
```

- Attach to to running sessions

```
1 tmux attach -t 0
```

- You need to login to the same login node (e.g., `levante3`<sup>1</sup>)

1. SSH config

# X11 forwarding

allows you to run graphical applications on a remote server

- Requires an X11-server on your local machine (e.g. [XQuartz](#))
- Pass the `-X`<sup>1</sup> option to your `ssh` command<sup>2</sup>

```
1 ssh -X levante
```

1. Mac users might want to use `-Y`, or MacOS will quit the forwarding after about 20 min.
2. or check the SSH config option



# Port forwarding (tunneling)

allows secure transmission of data over SSH

- Use `-L` to forward a local port to a port on the remote machine

```
1 ssh -L 8888:localhost:8888 levante # <local_port>:localhost:<remote_port>
```

- Data served on the remote port (e.g. JupyterHub, HTTP server) is then accessible locally at <http://localhost:8888>



# Further reading

- [Software Carpentry: The Unix Shell](#)
- [Software Carpentry Incubator: Unix Shell Extras \(incl. ssh\)](#)

