Operating Systems and Networks

Prof. Dr. Christoph Schober

Department of Applied Computer Science,
Deggendorf Institute of Technology

christoph.schober@th-deg.de

Beginning with Linux

Beginning with Linux

1. Motivation

Why Linux and not something else (e.g., Windows)?

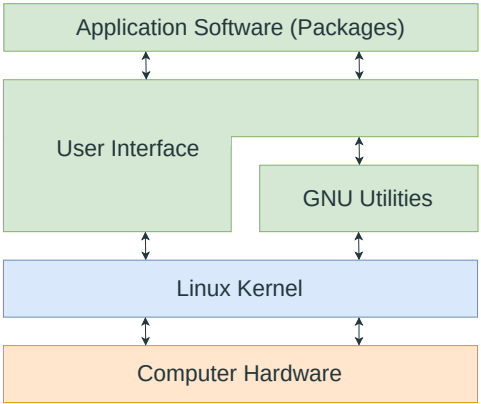
- All of the world's top 500 supercomputers run on Linux[1].
- 85% of all smartphones are powered by Linux[2].
- 96% of the world's top 1 million servers run on Linux[3].
- 90% of the public cloud runs on Linux[4].
- 47% of professional developers use Linux-based operating systems.[5]
- Linux is in the top 10 of essential skills a software engineer should have[6]

[1] Source: It's FOSS
[2] Source: Hayden James
[3] Source: ZDNET
[4] Source: Kernel Development Report
[5] Source: Truelist
[6] Source: Medium

A Linux system consists of four parts:

- The Linux kernel.
- A desktop environment.
- A set of GNU utilities.
- A set of software packages.

A Linux **distribution** is a specific combination of these components.



The Linux kernel was created[7] and is still maintained by **Linus Torvalds**:

[7] In the year 1991

Linux distributions differ in:

- Version of the Linux Kernel.
- Desktop environments (Gnome, KDE, Unity, Xfce, ...)
- Software package managers (apt, dnf, yum, pacman, ...)
- Software package update strategy (stable vs. bleeding edge)
- Display servers[8] (X11, wayland)
- Release model (cyclic, rolling, long-term support)
- Goals and aims (e.g, LibreELEC, a Linux distribution for media centers)
- Supported platforms (e.g., Raspbian for the Raspberry Pi)

[8] A program which is responsible for the input and output coordination of users.

Througout the course, we use **Ubuntu 22.04 LTS** in the lab:

- Kernel 5.15 LTS
- Gnome 42
- Apt package manager
- Wayland
- Recent software packages
- Cyclic releases (incl. LTS[9]).
- Aims to be the best desktop distribution on the market.

[9] Long-Term Support

Beginning with Linux

2. The Shell

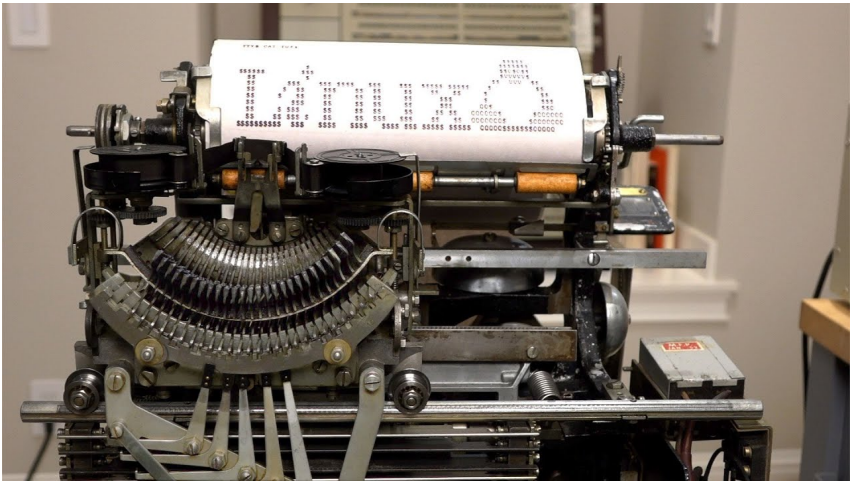
The Shell

A user interface for accessing an operating system's services, often providing a command-line interface for executing commands and scripts.

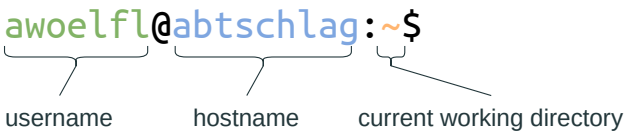
Accessing the shell:

- Many Linux distributions initiate multiple **virtual consoles**, accessible using a singular keyboard and monitor via shortcuts like (CTRL + ALT + F1-F7).
- Additionally, a majority of Linux distributions offer a **terminal emulator** application, which can be launched from the desktop environment.
- Shell commands can be inputted through the Linux **console**[10] mode.

[10] A term rooted in the era when hardwired console terminals were the standard for UNIX access.



After logging in to a virtual console, the *shell prompt* is shown:



Description:

- The name of the system is called **hostname**.
- The "~" sign for the working directory represents the users **home directory**.
- The default prompt symbol for the *Bash*[11] is the dollar sign **$**.

[11] Bourne-Again shell, written by Brian Fox for the GNU project. Default shell in Ubuntu.

Command

Program that is executed when a user enters its name at the shell prompt and presses Enter on the keyboard.

The basic syntax of command line programs is as follows[12]:

```
CommandName [Option(s)] [Argument(s)]
```

Example: The `echo` program displays the text provided as an argument

```
$ echo "Hello World"
Hello World
```

[12] Brackets indicate an optional part of the syntax.

Most Linux distributions include an online **manual** for looking up information on shell commmands, as well as lots of GNU utilities.

The `man` command followed by a specific command name as argument provides access to that utility's manual entry. Example:

```
1 $ man echo
```

→ Becoming familiar with the manual is invaluable for working with Linux.

One method for getting things done faster is using the command-line **history**. Typically, the last 1000 executed commands are kept in the history list.

The `history` command without options or arguments shows the history list:

```
1 $ history
```

Useful key combinations to interact with the history on the shell prompt:

- Arrow Up / Arrow Down: Navigate through the history list.
- CTRL + R: Search in the history list.

Assignment

Complete the task ”Shell Commands” on iLearn.

Environment Variable

A named value stored within the operating system that can affect the way running processes will behave on a computer.

Scope of environment variables:

- *Global:* Accessible across all shells and to all users.
- *User-defined*: Limited to the shell in which they were established.

Declare a user-defined environment variable[13]:

```
1 $ MY_FOO=bar  # MY_FOO is the variable name, BAR is its value
```

Declare a user-defined environment variable that is visible from subshells:

```
1 $ export MY_FOO=bar
```

Reading an environment variable:

```
1 $ echo $MY_FOO
```

Delete an user-defined environment variables:

```
1 $ unset MY_FOO
```

[13] Variable names should have all capital letters with underscores as seperators.

List all currently declared environment variable:

```
1 $ env
```

Useful global environment variables:

ENV	Description
USER	The name of the current user account.
HOME	The absolute path to the user's home directory.
PWD	The absolute path to the current working directory.
PATH	Colon-separated list where the shell searches for commands.
HOSTNAME	The name of the computer.

Beginning with Linux

3. Files

File

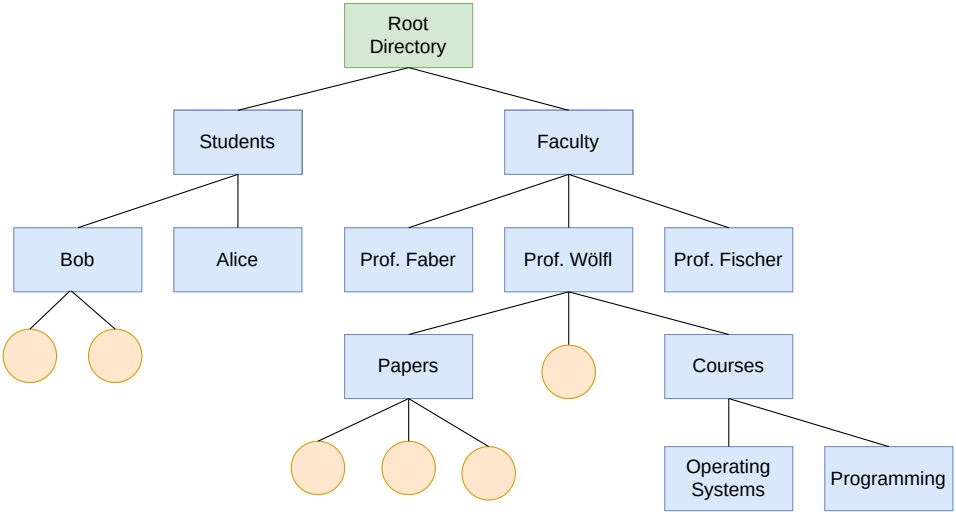
A digital container used to store data or information on a computer system.

Basic characteristics:

- Every file possesses a **name** and an **extension**[14].
- Files are organized hierarchically using **directories**.
- A file resides on a storage medium, such as a disk, CD-ROM, or USB drive.

→ Operating systems **obscure the intricacies** of the underlying storage devices.

[14] Typically in the format `filename.ext`, e.g. `document.txt`


Root Directory
Students
Faculty
Bob
Alice
Prof. Faber
Prof. Wölfl
Prof. Fischer
Papers
Courses
Operating Systems
Programming

Virtual Directory

A unified directory structure that aggregates file paths from all connected storage devices in the system.

Understanding file paths:

- The foundational directory is termed the *root directory*.
- Subdirectories and files under the root directory are represented by their specific directory paths[15].
- In a path, directories are delineated by a forward slash (`/`).

Example: `/home/awoelfl/documents/exam.pdf`

[15] This approach is analogous to Windows.

Absolute Path: A path that details the file location by enumerating directories from the root directory to the target file.

Example: `/home/awoelfl/documents/exam.pdf`

Relative Path: For each process, there exists a **working directory**[16]; path names not prefaced with a slash are sought within this directory.

Example: `documents/exam.pdf`

[16] Typically, this is the directory where the process initiated.

Useful commands when working with files and directories:

- `ls` (list directory content). Examples:

```
$ ls      # lists information about the files in the current directory
$ ls /    # lists information about the files in the root directory
$ ls -la  # enables long listing format and hidden files
```

Note: Files starting with `.` are considered hidden.

- `cd` (change directory). Examples:

```
$ cd /              # changes the working directory to /
$ cd home/woelfl    # changes the working directory to /home/woelfl
$ cd ..             # changes the working directory to /home
```

Useful commands when working with files and directories:

- `mkdir`, make directory. Examples

```
$ mkdir /tmp/mydir      # creates the directory /tmp/mydir
$ mkdir -p /tmp/my/dir  # creates the directory /tmp/my/dir,
                        # as well as parent directories as needed
```

- `rm`, remove directory. Example:

```
$ rm /tmp/file.txt      # removes a file
$ rm -r /tmp/my         # removes files or directories and
                        # their contents recursively
```

Useful commands when working with files and directories:

- `cat`, concatenate, Examples

```
1 $ cat /var/log/syslog       # prints the content of /var/log/syslog
2 $ cat file1.txt file2.txt   # prints the contents of both files
```

- `touch`. Examples:

```
1 $ touch myfile3.txt         # creates an empty file named myfile3.txt
```

Useful commands when working with files and directories:

- `mv`, move, Examples

```
1 $ mv file1.txt /tmp       # moves a file to a directory
2 $ mv file1.txt file2.txt  # renames a file
```

- `cp`, copy, Examples:

```
1 $ cp file1.txt /tmp       # places a copy of the file in a directory
2 $ cp file1.txt file2.txt  # creates a copy of the file with a new name
```

Assignment

Complete the task ”Files and Directories” on iLearn.

Fundamental Linux design principle: Everything is a file.

→ Refers to the idea that most I/O resources, including hardware devices, can be interacted with in a way that is consistent with how one would interact with files.

The Linux directory structure is standardized with the *Linux Filesystem Hierarchy Standard (FHS)* (1):

`/`	The root of the virtual directory. No files should be placed here.
`/bin`	The binary directory, where user-level utilities are stored.
`/lib`	The library directory, where program library files are stored.
`/boot`	The boot directory, where boot files are stored.
`/dev`	A virtual directory that represents hardware devices on the system.
`/proc`	A virtual directory that exposes system and process states.
`/sys`	A virtual directory used to configure the kernel and devices.

The Linux directory structure is standardized with the *Linux Filesystem Hierarchy Standard (FHS)* (2):

`/home`	The home directory, where Linux creates user directories.
`/root`	The root user account's Home directory.
`/etc`	The system configuration files directory.
`/opt`	The optional directory, often used to store optional software packages.
`/tmp`	The temporary directory, used for temporary work files.
`/usr`	The user-installed software directory.
`/var`	The variable directory, for files that change frequently (e.g. logs)

File Descriptor

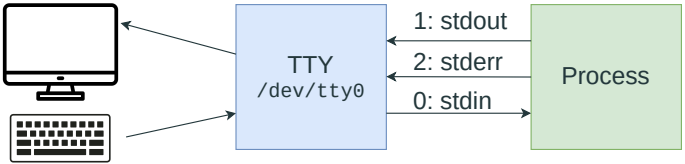
A unique integer identifier used to reference an open file or I/O resource.

By default, every process in Linux has three file descriptors open:

File Descriptor	Abbreviation	Description
0	`STDIN`	Standard Input
1	`STDOUT`	Standard Output
2	`STDERR`	Standard Error

→ These three special file descriptors are called the **standard streams** and handle the input and output of processes.

By default, the standard streams are connected to a terminal[17] device to gather inputs from the keyboard and print outputs to a display.



[17] `/dev/tty*` represent physical terminals or virtual consoles, whereas `/dev/pts/*` represent terminal emulator programs (pseudoterminals).

Redirect STDOUT to a file:

```
1 $ history > hist.txt
```

Append STDOUT to a file:

```
1 $ history >> hist.txt
```

Redirect STDERR to a file:

```
1 $ history 2> error.txt
```

Redirect both STDOUT and STDERR to a file:

```
1 $ history &> log.txt
```

Fundamental Linux design principles:

- Do one thing and do it well: Programs shall perform a single task very well.
- Text as universal interface: Programs shall use text as input and output.

→ This leads to chainability of programs allowing users to string together commands in powerful and often unforeseen combinations.

Pipe

A conduit allowing unidirectional flow of data between two processes.

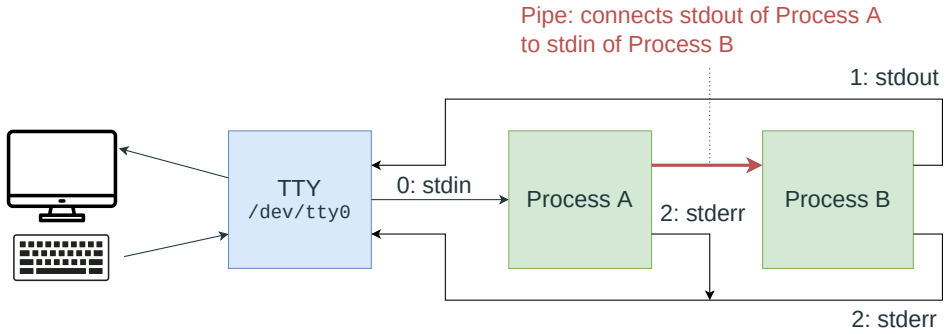
Understanding pipes:

- Established using the symbol `|`.
- Redirects the `STDOUT` of one process into the `STDIN` of another process.

Example:

```
1 $ history | sort -nr  # Sorts numerically and reverses order
```

Process A "piped" into process B:



Commands often used in conjunction with pipes:

- `less`, terminal pager, Example:

```
$ history | less
```

- `grep`, filter plain-text data, Example:

```
$ history | grep "ls"
```

- `awk`, extremely versatile text processing, Example:

```
$ history | awk '{print $1}'    # prints the first column
```

- `sed`, stream editor, Example:

```
$ history | sed "s/ls/whee/g"   # replaces 'ls' with 'whee'
```

Assignment

Complete the task ”Standard Streams” on iLearn.

Beginning with Linux

4. Users & Groups

User

An entity, either a person or an application, granted permissions to access and utilize system resources and services.

Characteristics:

- Users are authenticated using their *username* and corresponding *password*.
- A designated *home directory* exists for each user to house personal files:

```
1 /home/<username>
```

- Each user is uniquely represented by an integer, the *UID*.

The *root* user, with UID 0, possesses elevated privileges.

Group

A collection of users intended to simplify permission management by assigning shared rights to multiple users simultaneously.

Characteristics:

- Groups organize multiple users under a common label.
- Each group is uniquely represented by an integer, the *GID*.
- Default group for a user often shares the same name as their *username*.

Special groups, like *sudo*, grant users elevated privileges similar to the *root* user.

Privilege Delegation

The `sudo` command allows a system administrator to give certain users (or groups) the ability to run some (or all) commands as `root` user.

Example:

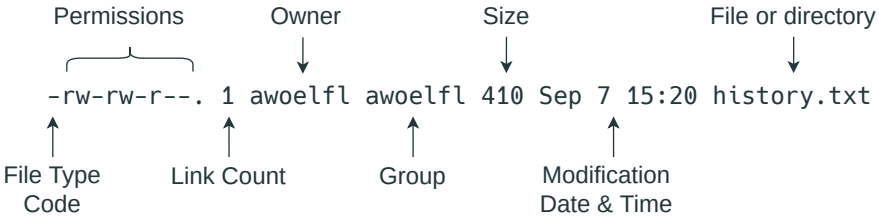
```
1 $ sudo shutdown -h now  # shuts down the machine now
```

Among others, the following tasks require root privileges:

- Adding, deleting or modifying user accounts.
- Installing system-wide software packages.
- Mounting storage devices.

File and directory permissions form the cornerstone of Linux's security model:

```
1 $ ls -l history.txt
```



Understanding **ownership:** Linux employs a three-tiered structure to secure its files and directories.

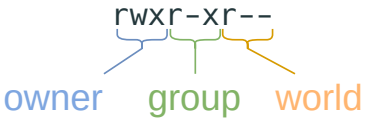
- **Owner**: Every file and directory is associated with a single user account. Permissions defined at this level are exclusively for the owner.
- **Group**: Files and directories are also linked to one group. The permissions here are applicable to all group members.
- **World**: These permissions cater to users who are neither the designated owner nor members of the associated group.

Ownership can be modified using the `chown` command. For example:

```
1 $ sudo chown -R awoelfl:staff /mnt/data/faculty # -R denotes recursive mode
```

Understanding **permissions:**

- **r** (read): Grants read access to a file.
- **w** (write): Allows modification of a file.
- **x** (execute): Provides execution rights for a file.



Modify permissions using `chmod` for owner (u), group (g), and others (o):

```
1 $ chmod u=rwx,g=rw,o=r file.txt  # set explicit permissions
2 $ chmod g+w file.txt             # add write permission for group
3 $ chmod g-w file.txt             # remove write permission from group
4 $ chmod +x file.txt              # grant execute permission for all
```

Commands to manage users and groups:

- `adduser`, create a new user, Example:

```
1 $ sudo adduser jsmith
```

- `addgroup`, create a new group, Example:

```
1 $ sudo addgroup students
```

- `usermod`, modify a user account, Example[18]:

```
1 $ sudo usermod -a -G sudo jsmith  # adds jsmith to the sudo group
```

[18] In many Debian based Linux distributions, the `sudo` group is preinstalled and its members are granted the permission to execute the `sudo` command.

Assignment

Complete the task ”Users & Groups” on iLearn.

Beginning with Linux

5. Package Management

Package

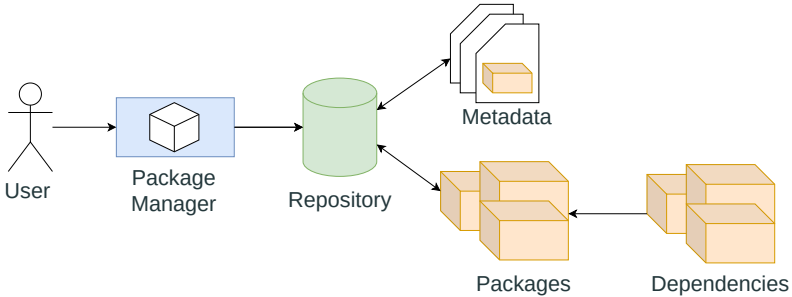
A bundled collection of files for a specific software application, library, or service, streamlined for installation and configuration on a Linux system.

A *Package Management System* ensures:

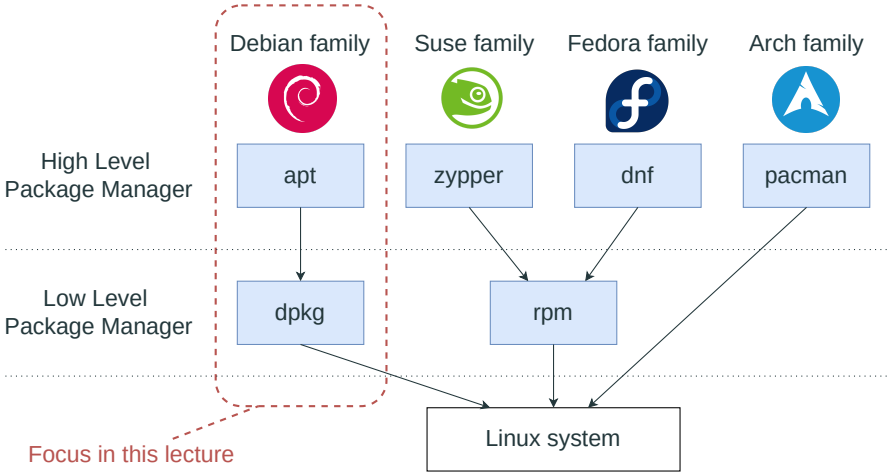
- A coherent record of software packages present on the system.
- Comprehensive tracking of files associated with each package.
- Up-to-date versions of installed software packages.

Note: Many software packages come with *dependencies* — additional packages that need to be installed prior or alongside them.

Software packages reside on servers, termed **repositories**. These are interfaced through package management utilities on the local Linux system:



Note: While metadata broadly refers to data characterizing other data, in this context, it provides insights about the packages within the repository.


Debian family
Suse family
Fedora family
Arch family
High Level
Package Manager
apt
zypper
dnf
pacman
Low Level
Package Manager
dpkg
rpm
Linux system
Focus in this lecture

Package management for Debian-based[19] systems[20]:

- The `dpkg` program is at the core of Debian-based package management.
- It provides options to install, update, and remove Debian package files.
- Assumes that the package file is locally available on the disk or directly downloadable from an URL.

→ To install application packages from the repository, the *Advanced Package Tool* (`apt`) is used (frontend to `dpkg`).

[19] That includes Ubuntu, Mint, Deepin, Kali, Raspbian,...

[20] Note that other distributions may have different package management systems.

Managing packages with *apt*:

- The basic format of the `apt` command is:

```
$ apt [options] command  # command defines the action to take
```

- List all installed packages on the system:

```
$ apt --installed list
```

- Display detailed information about a package:

```
$ apt show zsh  # zsh is the name of a software package
```

Managing packages with *apt*:

- Updating *package lists* (metadata) for new or upgradable packages:

```
$ sudo apt update
```

- Install a package:

```
$ sudo apt install zsh
```

- Remove a package:

```
$ sudo apt purge zsh
```

- Update all packages:

```
$ sudo apt upgrade
```

The *apt* repositories:

- Most Linux distributions ship with preconfigured software repositories.
- The default software repository locations are stored in:

```
1 /etc/apt/sources.list
```

- Non-standard software packages can be installed via *Personal Package Archive* (PPA), which are external repositories that can be added as follows:

```
1 $ sudo add-apt-repository ppa: name/ppa
```

Distribution agnostic systems:

- *Snap:*
  - Introduced by Canonical[21].
  - Packages encompass necessary libraries and dependencies.
- *Flatpak:*
  - Packages operate within a sandbox, ensuring isolation from the host system.
  - Bundles all prerequisites to execute the software.
- *AppImage:*
  - Follows a one-application-per-file paradigm.
  - Noteworthy for not necessitating sudo privileges.

[21] The organization responsible for Ubuntu.

Assignment

Complete the task ”Software Packages” on iLearn.

Beginning with Linux

6. Terminal Editors

The *vim* editor:

- The Primagean[22] uploaded a great video series on YouTube[23]
- The `vim` command starts the editor.

```
1 $ vim file.txt
```

- The `vim` editor has three standard modes:
  - *Normal Mode*: Used to navigate and perform basic operations.
  - *Insert Mode*: Used to edit and manipulate text.
  - *Visual Mode*: Used to select chunks of text to be manipulated.
- Use `esc` to fall back normal mode (from any other mode).

[22] https://www.youtube.com/c/ThePrimeagen
[23] https://www.youtube.com/playlist?list=PLm323Lc7iSW_wuxqmKx_xxNtJC_hJbQ7R

The *vim* editor:

- Perform simple text editing:

Keystroke	Description
`i`	Switch to insert mode

- Save & exit the editor (switch to normal mode first):

Keystroke	Description
`:w`	Save (write) the current changes to the disk.
`:q`	Exit the editor
`:wq`	Save and exit the editor

The *nano* editor:

- In contrast to `vim`, which is complicated but powerful, `nano` is simple.
- The `nano` command starts the editor.

```
1 $ nano file.txt
```

- Easy navigation with arrow keys.
- No change of modes required to perform text editing.
- The keystroke `CTRL+X` saves and exists the editor.

Beginning with Linux

7. Scripting

Shell Script

A text file that contains a sequence of commands for the shell to execute.

Benefits of scripting:

- Very efficient for repetitive tasks.
- For tasks requiring multiple commands, a script can streamline the process.
- Rule of thumb: If a task is repeated more than 3 times, consider scripting it.

A simple shell script:

```
1 #!/bin/bash
2 # This sample script displays the date/time
3 # and who is logged onto the system
4 date
5 who
```

Explanation:

- The first line is called the *shebang* (#!) and tells the current shell which shell to use for running the script.
- Lines 2-3 are comments (start with #) and are not interpreted by the shell.
- Lines 4-5 contain the commands that are executed in sequence.

Running shell scripts:

- Add permission to execute:

```
$ chmod +x script.sh
```

- Execute the script:

```
$ ./script.sh
$ bash script.sh
```

Best practice: Use the `.sh` file extension (even for bash scripts).

A script demonstrating user-defined variables:

```
1 #!/bin/bash
2 # Trying out user variables
3 days=10
4 guest="London Ratford"
5 echo "$guest checked in $days ago."
6 days=5
7 guest="Takoda Puddle"
8 echo "$guest checked in $days ago."
```

Explanation:

- Lines 3,4,7,8 define user variables.
- Lines 5,9 print text with user variables.

A script demonstrating functions:

```
1 #!/bin/bash
2 function greet() {
3     echo "Hello $1"
4 }
5 greet "Kai"
6 greet "Ben"
```

Explanation:

- No parameter declaration in the function's signature.
- *Positional parameters*: Special variables (`$0`, `$1`, `$1`, ..., `$n`) that hold the values of arguments passed, with `$0` representing the script's name.

A script demonstrating positional parameters:

```
1 #!/bin/bash
2 # Trying out command-line parameters
3 echo "Hello $1, glad to meet you.
```

Explanation:

- Assume that the script is executed as follows:

```
1 $ ./script.sh "Andreas"
2 Hello Andreas, glad to meet you."
```

Further reading:

- Shell scripts go way beyond just repeating commands.
- They provide control structures such as *if-then-else* and *loops*.

→ Once you are familiar with control structures, read about advanced shell scripting in [3], Chapter 19.

Assignment

Complete the task ”Scripting” on iLearn.

Beginning with Linux

8. Summary

Summary

You should have acquired the competencies to (1)

- Define and explain the newly introduced technical terms.
- Name and describe the parts of a Linux operating system.
- Use and describe the newly introduced commands on a Linux shell.
- Self-educate on commands using built-in manuals.
- Declare and access environment variables.
- State the most common global environment variables.

Summary

You should have acquired the competencies to (2)

- Create, delete and modify files and directories.
- Understand and apply permissions on files and directories.
- Redirect the outputs of a command to a file.
- Chain multiple commands together using pipes.
- Manage users and groups on Linux systems.
- Install, update, and delete packages on Debian-based systems.
- Edit text with a terminal editor.
- Write basic shell scripts.