**Live Session II**

Special Commands

Password adjustment

adjust file for < 8 chars

**sudo nano /etc/security/pwquality.conf**

look for line

**minlen = 1**

**sudo passwd**

----

[**Dash to dock**](https://extensions.gnome.org/extension/307/dash-to-dock/) **install for GNOME site like for Ubuntu, Fedora etc. Desktop icons**

----

Emacs colorized editor!

**sudo apt-get install emacs**

start a file

**emacs hello.cpp**

Saving a file

**C-x C-s**

Exiting the editor

**C-x C-c**

----

Date/Time changes

**date**

To change your system time from PDT (Pacific Daylight Time) to CST (Central Standard Time) for Chicago

First, let's see the available time zones for Chicago:

**timedatectl list-timezones | grep Chicago**

You should see America/Chicago in the output.

To set the time zone to Chicago (CST/CDT), use the following command:

**sudo timedatectl set-timezone America/Chicago**

Verify the change:

**timedatectl**

This should show you the new time zone setting. Also check the current time again

**Date**

**----**

**Lynx**

Lynx is a text-based web browser for Linux and other Unix-like operating systems. Here are the key points about Lynx:

-It is one of the oldest web browsers still in use, originally developed in 1992.

-Lynx is a text-only browser that displays web pages in plain ASCII text, without images or multimedia content.

-It runs in the terminal/command-line interface, making it useful for systems without graphical displays or for users who prefer text-based interfaces.

-Lynx is known for being fast and lightweight, using minimal system resources compared to graphical browsers.

-It supports various internet protocols including HTTP, HTTPS, FTP, and Gopher.

-Lynx is particularly useful for:

Accessing websites on low-bandwidth connections

Browsing on older hardware with limited resources

Testing web pages for accessibility

Automated web scraping and scripting tasks

-Navigation in Lynx is done using keyboard commands, with arrow keys used to move between links.

It can be installed on most Linux distributions using package managers. For example, on Ubuntu.

Exercise - To install Lynx on Ubuntu and run it to open Google.com, follow these steps:

## Installation

1. Update your package list and install Lynx using the following commands:

**sudo apt update**

**sudo apt install lynx -y**

This will install the Lynx browser on your Ubuntu system.

**## Running Lynx**

2. To open Google.com using Lynx, simply enter the following command in your terminal:

**lynx google.com**

This command will launch Lynx and load the Google homepage in text mode.

**## Navigation Tips for Lynx**

**3. Once Lynx is open:**

**- Use the arrow keys to navigate between links**

**- Press Enter to follow a link**

**- Press Q to quit the browser**

**- Press G to enter a new URL[2][4]**

Remember that Lynx is a text-based browser, so it won't display images or multimedia content. It's designed for speed and low resource usage, making it ideal for certain tasks or for use on systems with limited capabilities.

Chrony for Ubuntu, Fedora etc

Simply stated, cron jobs use NTP (Network Time Protocol) is a networking protocol used to synchronize computer clocks over a network.

Key points about NTP:

Purpose: It ensures that all computers on a network have their clocks synchronized to the same time.

Accuracy: NTP can typically maintain time to within a few milliseconds of Coordinated Universal Time (UTC).

Hierarchy: It uses a hierarchical system of time sources, with the most accurate clocks at the top (stratum 1) and less accurate ones in lower strata.

Client-Server Model: Computers running NTP clients periodically check with NTP servers to update their time.

Wide Use: It's used across the internet and in many private networks to keep devices time-synchronized.

Continuous Operation: NTP typically runs as a background process, constantly making small adjustments to keep the system clock accurate.

Security: It helps maintain the integrity of time-sensitive operations in computing, such as financial transactions or log file timestamps.

In essence, NTP is the internet's way of ensuring that all connected devices agree on what time it is, which is crucial for many aspects of network communication and system operations.

To install **Chrony** using APT in Ubuntu, you can follow these steps:

**1. Update the package list** to ensure you're installing the latest version available:

**sudo apt update**

**2. Install Chrony**:

**sudo apt install chrony**

**3. Enable and start the Chrony service** (if it's not automatically started):

**sudo systemctl enable chrony**

**sudo systemctl start chrony**

**4. Check the status of Chrony** to verify that it's running:

**sudo systemctl status chrony**

Once installed, you can use chronyc tracking to check the time synchronization status.

Exercise – set up shell script and cron job to execute shell script after a minute has lapsed.

To set up a cron job in Ubuntu that triggers a bash shell script after 1 minute, follow these steps:

1. **\*\*Create the Bash Script\*\***

First, create a simple bash script that you want to trigger with the cron job.

For example, create a script called `my\_script.sh` that logs the current date and time to a file.

**#!/bin/bash**

**# my\_script.sh - Logs current date and time to a file**

**echo "Script executed at: $(date)" >> /home/yourusername/script\_log.txt**

Make the script executable:

**chmod +x /home/yourusername/my\_script.sh**

2. **\*\*Edit the Cron Table\*\***

Open the cron table for editing by running:

**crontab -e**

This will open the crontab file in the default text editor. If this is your first time using **crontab**, it will prompt you to choose an editor (you can select `nano` for simplicity).

3. **\*\*Add a Cron Job\*\***

Add the following line at the end of the file to schedule the script to run after 1 minute:

**\* \* \* \* \* /home/yourusername/my\_script.sh**

This cron timing `**\* \* \* \* \***` means the script will run every minute. Here's the breakdown of cron syntax:

```

\* \* \* \* \*

- - - - -

| | | | |

| | | | +---- day of the week (0 - 7) (Sunday = 0 or 7)

| | | +------ month (1 - 12)

| | +-------- day of the month (1 - 31)

| +---------- hour (0 - 23)

+------------ minute (0 - 59)

```

4. **\*\*Save the Cron Job\*\***

After adding the cron job, save the file and exit the editor (e.g., for `nano`, press `CTRL + X`, then `Y`, and hit `Enter`).

5. **\*\*Verify the Cron Job\*\***

To verify that the cron job is set, you can list all the cron jobs with:

**crontab -l**

6. **\*\*Wait and Check the Logs\*\***

After 1 minute, the script should run, and you should see the output in the file `/home/yourusername/script\_log.txt`.

You can check the log by running:

**cat /home/yourusername/script\_log.txt**

Each time the script runs, it will append the current date and time to this file.

View the current time sources:

**chronyc sources**

Check the synchronization status:

**chronyc tracking**

[ To manually synchronize the system clock: ]

**sudo chronyc makestep**

To see detailed measurements from each NTP source:

**chronyc sourcestats**

Chrony's default servers:

By default, Chrony doesn't always use the same servers. The default configuration in Fedora typically includes a pool of NTP servers, often from the Fedora NTP pool.

When Chrony is active and running on your Fedora system, it uses NTP (Network Time Protocol) servers to maintain accurate time settings.

Here's a more detailed explanation:

Continuous Synchronization: Chrony constantly communicates with the configured NTP servers to keep your system time accurate.

Automatic Adjustments: It makes small, gradual adjustments to your system clock to keep it synchronized with the NTP servers, which are typically very accurate time sources.

Multiple Sources: Chrony usually uses multiple NTP servers for redundancy and improved accuracy. It can determine which sources are most reliable.

Handles Network Issues: Chrony is designed to work well even with intermittent network connections, making it suitable for various environments.

Precision: It can achieve sub-millisecond accuracy on LANs and up to a few milliseconds over the Internet.

Time Zone Independence: NTP always deals with UTC (Coordinated Universal Time). Your system's time zone settings (like CST for Chicago) are applied on top of this accurate UTC time.

Drift Compensation: Chrony learns how your system clock drifts over time and can compensate for this, improving accuracy even when temporarily disconnected from NTP servers.

Gradual Corrections: For large time discrepancies, Chrony can be configured to correct the time gradually to avoid negatively impacting system processes.

\*So, with Chrony active, your Fedora system is indeed using NTP servers to maintain very accurate time settings. The local time you see (like CST for Chicago) is then derived from this accurate UTC time based on your system's time zone settings.

Benefits of running your own NTP server, and nodes connected to it can sync their time with it. Here's a summary of how this works:

Setting up an NTP server:

Choose a computer to act as your NTP server.

Install NTP software (like ntpd or chrony) on this machine.

Configure it to sync with reliable external time sources.

Enable the NTP service to run as a server.

Syncing nodes to your NTP server:

On each client node, install an NTP client (again, ntpd or chrony).

Configure the client to use your NTP server as its time source.

This is typically done by editing the NTP configuration file (e.g., /etc/ntp.conf or /etc/chrony.conf).

Add a line like: "server your\_ntp\_server\_ip"

Benefits of running your own NTP server:

Reduced external network traffic, as only one machine needs to query external time sources.

Consistent time across your local network.

Better control over time synchronization for your network.

Best practices:

For larger networks, consider having 2-3 local NTP servers for redundancy.

Ensure your NTP server has a stable and accurate time source.

Regularly monitor the NTP server's status and the synchronization of client nodes.

Synchronization process:

Client nodes will periodically check with your NTP server to adjust their time.

The NTP protocol handles the complexities of calculating time offsets and gradually adjusting system clocks.

Running your own NTP server can provide more consistent and controlled time synchronization across your network, especially for environments with many devices or limited internet connectivity

----

Working with curl commands

cURL (Client URL) is a command-line tool and library for transferring data using various network protocols. Here are the key points about cURL:

1. Purpose: It's used to transfer data to and from a server, supporting many protocols including HTTP, HTTPS, FTP, and more.
2. Command-line interface: cURL provides a CLI for making network requests directly from the terminal.
3. Versatility: It supports a wide range of operations, including downloading files, API testing, and network diagnostics.
4. Cross-platform: cURL is available on multiple operating systems, including Windows, macOS, and Linux.
5. Protocols: It supports numerous protocols, making it a versatile tool for various networking tasks.
6. Options and customization: cURL offers many command-line options to customize requests, including setting headers, using authentication, and handling cookies.
7. Scripting and automation: It can be easily integrated into scripts for automation tasks.
8. API interaction: cURL is commonly used for testing and interacting with APIs.
9. Debugging: It's useful for troubleshooting network issues and inspecting server responses.
10. Data formats: cURL can handle various data formats, including JSON and form data.
11. Security: It supports SSL/TLS for secure communications.
12. Library (libcurl): Besides the command-line tool, cURL also provides a library that can be used in programming languages for making network requests.

cURL is widely used by developers, system administrators, and network professionals for various tasks related to data transfer and API interactions. Its simplicity and power make it a valuable tool in many networking and web development scenarios.

\*Sample curl commands

Create tmp directory and travel (cd) to it

For text file

**curl -O** [**https://www.gutenberg.org/files/1342/1342-0.txt**](https://www.gutenberg.org/files/1342/1342-0.txt)

Observe content of txt file with cat command

**cat 1342-0.txt**

Do a word count

**wc -l 1342-0.txt**

For html file (download with different file name)

**curl -o myfile.html https://www.example.com/index.html**

For image file

**curl -O** [**https://upload.wikimedia.org/wikipedia/commons/thumb/3/35/Tux.svg/1200px-Tux.svg.png**](https://upload.wikimedia.org/wikipedia/commons/thumb/3/35/Tux.svg/1200px-Tux.svg.png)

Open image file via command file above via ImageMagick!

**display 1200px-Tux.svg.png**

\*if necessary install imageMagick

**Dockerization Containers**

Docker (for httpd/php/mysql services, etc.)

Docker simplifies the process of application deployment by packaging software and its dependencies into containers. These containers are lightweight, portable, and can run consistently across *different* computing environments, **ensuring** that applications behave the same way regardless of where they are deployed. This capability is particularly valuable in modern software development, where applications are often distributed across various platforms and infrastructure.

Containers vs. Virtual Machines

Containers are often compared to virtual machines (VMs), but they operate differently. While VMs virtualize the entire hardware stack, containers share the host operating system's kernel and utilize its resources more efficiently. This results in faster startup times and lower overhead, allowing multiple containers to run on a single host without the resource constraints typically associated with VMs

Components of Docker

Docker consists of several key components:

* **Docker Engine**: The core service that runs and manages containers.
* **Docker Images**: Read-only templates used to create containers. They include everything needed to run an application, such as code, libraries, and environment variables.
* **Docker Containers**: Instances of Docker images that run the application in an isolated environment.
* [**Docker Hub**](https://hub.docker.com/): A cloud repository where users can share and access Docker images

Benefits of Using Docker

1. **Portability**: Docker containers can run on any system that supports Docker, making it easy to move applications between development, testing, and production environments.
2. **Efficiency**: Containers share the host OS kernel, reducing resource consumption compared to traditional VMs. This allows for higher density of applications on a single host.
3. **Speed**: Containers can be started and stopped quickly, facilitating rapid deployment and scaling of applications.
4. **Isolation**: Each container runs in its own environment, which enhances security and stability by preventing applications from interfering with one another

Let’s do it!

Here's the adjusted version of the commands for Docker on Ubuntu:

Install Docker:

# Update package index

**sudo apt-get update**

# Install prerequisites

**sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common**

# Add Docker's official GPG key

**curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -**

# Add Docker repository

**sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"**

# Install Docker Engine

**sudo apt-get update**

**sudo apt-get install -y docker-ce docker-ce-cli containerd.io**

Start service, enable on boot:

**sudo systemctl start docker**

**sudo systemctl enable docker**

Create a Docker Container with httpd, PHP, and MySQL:

Create a `docker-compose.yml` file:

**sudo nano docker-compose.yml**

Add the following content:

version: '3.8'

services:

web:

image: nginx:latest

ports:

- "80:80"

db:

image: mysql:latest

environment:

MYSQL\_ROOT\_PASSWORD: rootpassword

MYSQL\_DATABASE: mydatabase

MYSQL\_USER: user

MYSQL\_PASSWORD: userpassword

volumes:

- db\_data:/var/lib/mysql

volumes:

db\_data:

Create a Simple PHP Page:

**mkdir html**

**cd html**

**sudo nano index.php**

Add the following content to **index.php**:

**<?php**

**phpinfo();**

**?>**

Run Docker Compose:

**sudo docker-compose up -d**

View web page:

**curl localhost:80**

Create a Database and Table in MySQL:

# View running containers

**sudo docker ps**

# Access the MySQL container

**sudo docker exec -it <your\_container\_id\_of\_db> mysql -u root -p**

# Inside the MySQL shell, execute:

**CREATE DATABASE testdb;**

**USE testdb;**

**CREATE TABLE users (id INT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255), email VARCHAR(255));**

**INSERT INTO users (name, email) VALUES ('John Doe', 'john@example.com'), ('Jane Doe', 'jane@example.com');**

**SELECT \* FROM users;**