**بسم الله الرحمن الرحیم**

**فاطمه اثباتی**

**هدف پروژه**

هدف از این پروژه، ایجاد یک linux shell script است که فرایند log rotation را در محیط لینوکس شبیه سازی کند. این اسکریپت باید فایلی را که در اختیارتان قرار میگیرد، از انتها به صورت خط به خط بخواند و بر اساس معیارهای ذکر شده در فایل های جداگانه سازماندهی کند.

**linux shell script چیست؟**

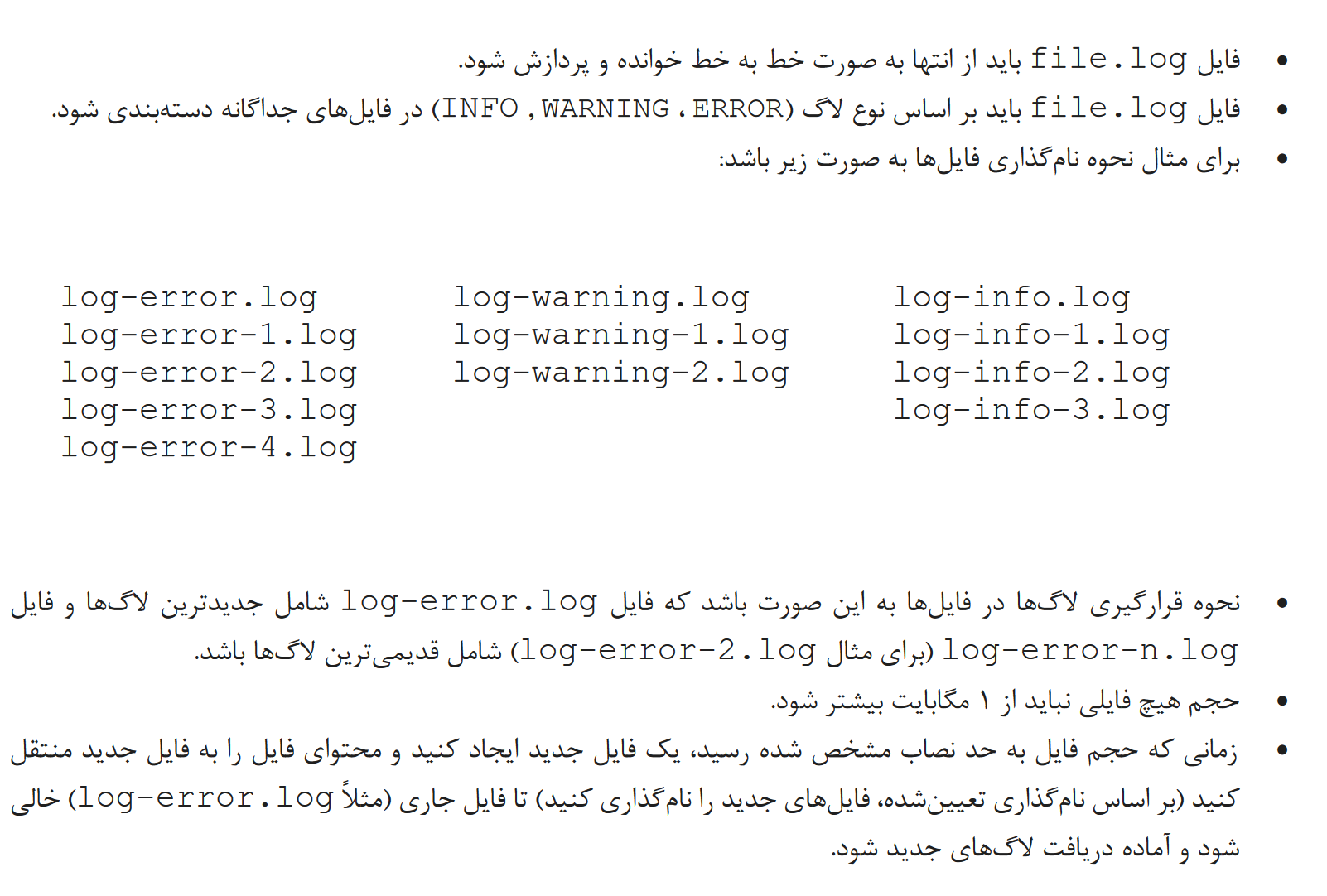
Linux Shell Script یک برنامه یا فایل متنی است که حاوی مجموعه‌ای از دستورات است که در محیط خط فرمان (Shell) سیستم عامل لینوکس یا دیگر سیستم‌های یونیکس-مانند اجرا می‌شود. این اسکریپت‌ها به برنامه‌نویسان و کاربران اجازه می‌دهند تا وظایف مختلف را به‌صورت خودکار انجام دهند. به عبارت دیگر، Shell Scripting راهی برای نوشتن و اجرای دستوراتی است که به صورت متوالی و بدون نیاز به مداخله انسانی انجام می‌شوند.

Shell چیست؟

Shell در سیستم‌عامل‌های لینوکس یک واسط کاربری است که بین کاربر و هسته(Kernel) سیستم‌عامل قرار دارد. این واسط به کاربر اجازه می‌دهد دستورات را وارد کرده و با سیستم عامل ارتباط برقرار کند. مشهورترین شل‌ها عبارتند از:

* Bash (Bourne Again Shell) محبوب‌ترین شل در لینوکس.
* Zsh یک شل پیشرفته که امکانات بیشتری نسبت به Bash ارائه می‌دهد.
* Ksh (Korn Shell) و Csh (C Shell) نیز از دیگر شل‌های پرکاربرد هستند.

**قسمت اول:**



#!/bin/bash

# Define the main log file and the categorized log files

LOG\_FILE="file.log"

INFO\_LOG="log-info.log"

WARNING\_LOG="log-warning.log"

ERROR\_LOG="log-error.log"

# Define the maximum size for each log file (1 MB in bytes)

MAX\_SIZE=1048576

# Function to rotate log files if they exceed the maximum size

rotate\_log() {

    local log\_file=$1

    local base\_name=$(basename "$log\_file" .log)  # Get the base name of the log file

    local count=1

    # Find the next available suffix for the rotated log file

    while [ -f "${base\_name}-${count}.log" ]; do

        ((count++))

    done

    # Move the current log file to a new file with the incremental suffix

    mv "$log\_file" "${base\_name}-${count}.log"

    touch "$log\_file"  # Create a new empty log file

}

# Function to process the main log file and categorize logs

process\_log() {

    local log\_type=$1

    local log\_file=$2

    # Read the main log file from the end using tac and filter lines based on the log type

    while IFS= read -r line; do

        echo "$line" >> "$log\_file"  # Append the line to the corresponding log file

        if [ $(stat -c%s "$log\_file") -gt $MAX\_SIZE ]; then  # Check if the log file exceeds the maximum size

            rotate\_log "$log\_file"  # Rotate the log file if it exceeds the maximum size

        fi

    done < <(tac "$LOG\_FILE" | grep "$log\_type")  # Filter lines based on the log type

}

# Process INFO logs

process\_log "INFO" "$INFO\_LOG"

# Process WARNING logs

process\_log "WARNING" "$WARNING\_LOG"

# Process ERROR logs

process\_log "ERROR" "$ERROR\_LOG"

حالا جزء به جزء اسکریپتی که نوشته شده را بررسی میکنیم:

1) تعریف فایل‌های لاگ اصلی و لاگ‌های دسته‌بندی‌شده:

LOG\_FILE="file.log"

INFO\_LOG="log-info.log"

WARNING\_LOG="log-warning.log"

ERROR\_LOG="log-error.log"

**LOG\_FILE:** نام فایل اصلی لاگ که همه‌ی پیام‌ها در آن ثبت شده‌اند (در این مثال file.log).  
**INFO\_LOG**: نام فایلی که لاگ‌های مربوط به نوع INFO در آن ذخیره می‌شوند.  
**WARNING\_LOG:** نام فایلی که لاگ‌های مربوط به نوع WARNING در آن ذخیره می‌شوند.  
**ERROR\_LOG**: نام فایلی که لاگ‌های مربوط به نوع ERROR در آن ذخیره می‌شوند.

2) تعریف حداکثر اندازه برای هر فایل لاگ:

MAX\_SIZE=1048576

**MAX\_SIZE**: مقدار حداکثر اندازه فایل‌های لاگ که در اینجا برابر با 1 مگابایت (1048576 بایت) تعیین شده است. اگر اندازه فایل از این مقدار بیشتر شود، لاگ جدید ایجاد می‌شود و لاگ قبلی با استفاده از تابع rotate\_log چرخانده (rotated) می‌شود.

3) تعریف تابع چرخش (Rotate) فایل‌های لاگ:

rotate\_log() {

    local log\_file=$1

    local base\_name=$(basename "$log\_file" .log)

    local count=1

    while [ -f "${base\_name}-${count}.log" ]; do

        ((count++))

    done

    mv "$log\_file" "${base\_name}-${count}.log"

    touch "$log\_file"

}

**rotate\_log**: این تابع برای چرخش فایل‌های لاگ استفاده می‌شود و در صورتی که اندازه فایل لاگ از مقدار حداکثر تعریف شده بیشتر شود، یک فایل جدید با شماره‌گذاری ایجاد می‌کند.

* **local log\_file=$1**: این خط، نام فایل لاگ را از ورودی تابع دریافت کرده و آن را به متغیر log\_file اختصاص می‌دهد.
* **basename "$log\_file" .log**: این دستور از فایل لاگ، نام فایل بدون پسوند .log را برمی‌گرداند. به عنوان مثال، اگر ورودی log-info.log باشد، خروجی log-info خواهد بود.
* **count=1**: مقدار اولیه برای شمارش فایل‌های چرخانده‌شده است.
* **while [ -f "${base\_name}-${count}.log" ]**: این حلقه بررسی می‌کند که آیا فایل چرخانده‌شده‌ای با این شمارش وجود دارد یا خیر. اگر وجود داشت، شمارنده را افزایش می‌دهد تا شماره‌ی جدیدی پیدا کند که موجود نباشد.
* **mv "$log\_file" "${base\_name}-${count}.log"**: فایل لاگ موجود را با استفاده از دستور mv به یک فایل جدید با شماره‌گذاری جدید منتقل می‌کند.
* **touch "$log\_file"**: این دستور یک فایل لاگ خالی با همان نام اولیه ایجاد می‌کند تا بتوان از آن برای ذخیره لاگ‌های جدید استفاده کرد.

4) تعریف تابع برای پردازش و دسته‌بندی لاگ‌ها:

process\_log() {

    local log\_type=$1

    local log\_file=$2

    while IFS= read -r line; do

        echo "$line" >> "$log\_file"

        if [ $(stat -c%s "$log\_file") -gt $MAX\_SIZE ]; then

            rotate\_log "$log\_file"

        fi

    done < <(tac "$LOG\_FILE" | grep "$log\_type")

}

**process\_log**: این تابع وظیفه دارد که لاگ‌ها را از فایل اصلی بخواند و بر اساس نوع لاگ   
(INFO، WARNING، ERROR) آن‌ها را در فایل‌های جداگانه ذخیره کند. در عین حال، اگر اندازه هر فایل از حداکثر اندازه تعیین شده بیشتر شد، آن فایل را می‌چرخاند.

* **local log\_type=$1**: نوع لاگ (مانند INFO، WARNING، ERROR) را از ورودی تابع دریافت می‌کند.
* **local log\_file=$2**: نام فایل لاگی که باید لاگ‌های مربوط به نوع خاص را در آن ذخیره کند، از ورودی تابع دریافت می‌شود.
* **tac "$LOG\_FILE" | grep "$log\_type"**: این دستور ابتدا فایل لاگ اصلی را از انتها می‌خواند (با استفاده از دستور tac که برعکس cat است) و سپس خطوطی که با نوع لاگ مشخص‌شده مطابقت دارند (با استفاده از grep) فیلتر می‌کند. دلیل استفاده از tac، این است که اسکریپت خطوط جدیدتر را ابتدا پردازش کند.
* **IFS= read -r line**: دستور read به طور خط به خط فایل لاگ فیلترشده را می‌خواند. IFS= به این معنی است که فاصله‌ها و تب‌ها به عنوان جداکننده‌های خطوط در نظر گرفته نمی‌شوند. -r هم تضمین می‌کند که بک‌اسلش‌ها (\) نادیده گرفته نمی‌شوند.
* **echo "$line" >> "$log\_file"**: خط فیلتر شده را به فایل لاگ مربوطه اضافه می‌کند.
* **stat -c%s "$log\_file"**: این دستور اندازه فایل لاگ را بر حسب بایت برمی‌گرداند. اگر اندازه فایل از مقدار MAX\_SIZE بیشتر شد، فایل چرخانده می‌شود.

5) پردازش لاگ‌ها برای هر نوع لاگ:

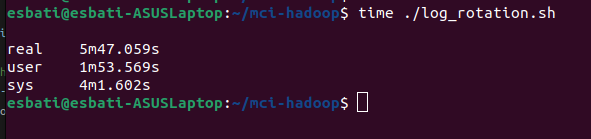
process\_log "INFO" "$INFO\_LOG"

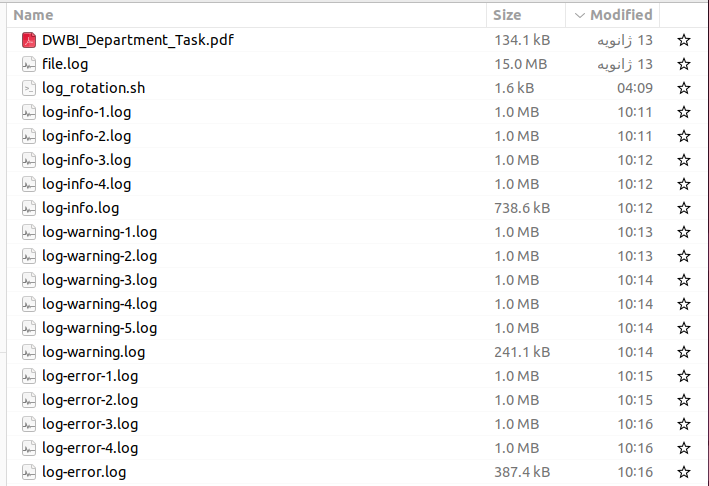
process\_log "WARNING" "$WARNING\_LOG"

process\_log "ERROR" "$ERROR\_LOG"

این بخش سه بار تابع process\_log را فراخوانی می‌کند تا لاگ‌های مربوط به هر نوع (INFO، WARNING، ERROR) را پردازش کند.

* **process\_log "INFO" "$INFO\_LOG"**: لاگ‌های نوع INFO را از فایل اصلی می‌گیرد و در فایل log-info.log ذخیره می‌کند.
* **process\_log "WARNING" "$WARNING\_LOG"**: لاگ‌های نوع WARNING را از فایل اصلی می‌گیرد و در فایل log-warning.log ذخیره می‌کند.
* **process\_log "ERROR" "$ERROR\_LOG"**: لاگ‌های نوع ERROR را از فایل اصلی می‌گیرد و در فایل log-error.log ذخیره می‌کند.

****



**Hadoop چیست؟**

Hadoop یک فریم‌ورک متن‌باز است که برای ذخیره‌سازی و پردازش حجم عظیمی از داده‌ها به کار می‌رود. این فریم‌ورک توسط Apache Software Foundation توسعه داده شده و طراحی شده تا داده‌ها را به صورت توزیع‌شده و موازی بر روی چندین سرور یا سیستم پردازش کند.

Hadoop به‌طور کلی از چهار بخش اصلی تشکیل شده است:

1. Hadoop Distributed File System (HDFS): یک سیستم فایل توزیع‌شده که داده‌ها را به بلوک‌های کوچکی تقسیم می‌کند و آن‌ها را در سرورهای مختلف (nodes) ذخیره می‌کند. این سیستم از قابلیت تحمل خطا(اشکال پذیری) برخوردار است، به این معنا که در صورت از دست رفتن یک نود، نسخه‌های دیگری از داده وجود دارد.

2. MapReduce: یک مدل برنامه‌نویسی برای پردازش موازی داده‌ها در سرورهای مختلف. MapReduce به دو بخش تقسیم می‌شود:

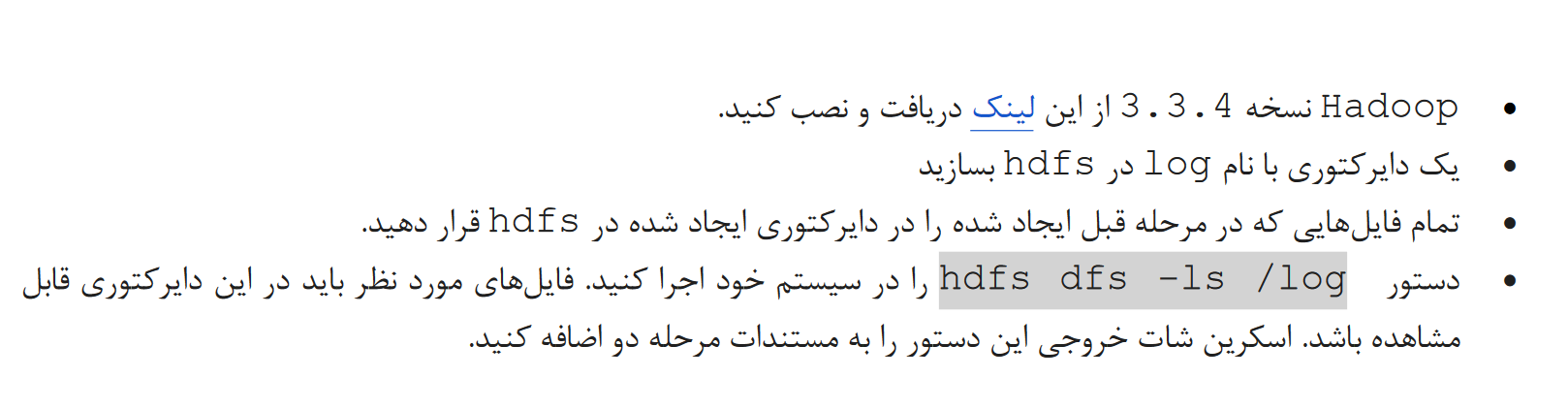
* Map: داده‌ها را به قسمت‌های کوچکی تقسیم و پردازش می‌کند.
* Reduce: نتایج پردازش شده را جمع‌بندی و تجمیع می‌کند.

3. YARN (Yet Another Resource Negotiator): بخشی از Hadoop که مدیریت منابع و برنامه‌های کاربردی را در سیستم‌های توزیع‌شده انجام می‌دهد.

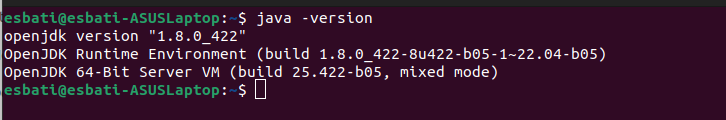
4. Hadoop Common: مجموعه‌ای از ابزارها و کتابخانه‌های عمومی که از دیگر بخش‌های Hadoop پشتیبانی می‌کنند.

Hadoop به دلیل مقیاس‌پذیری بالا و توانایی مدیریت داده‌های بسیار بزرگ (Big Data)، به‌ویژه در حوزه‌هایی مثل تحلیل داده‌ها، داده‌کاوی و یادگیری ماشین، به‌طور گسترده مورد استفاده قرار می‌گیرد.

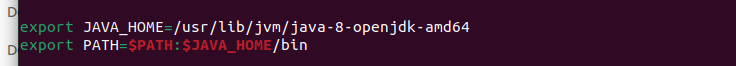
**قسمت دوم:**

****

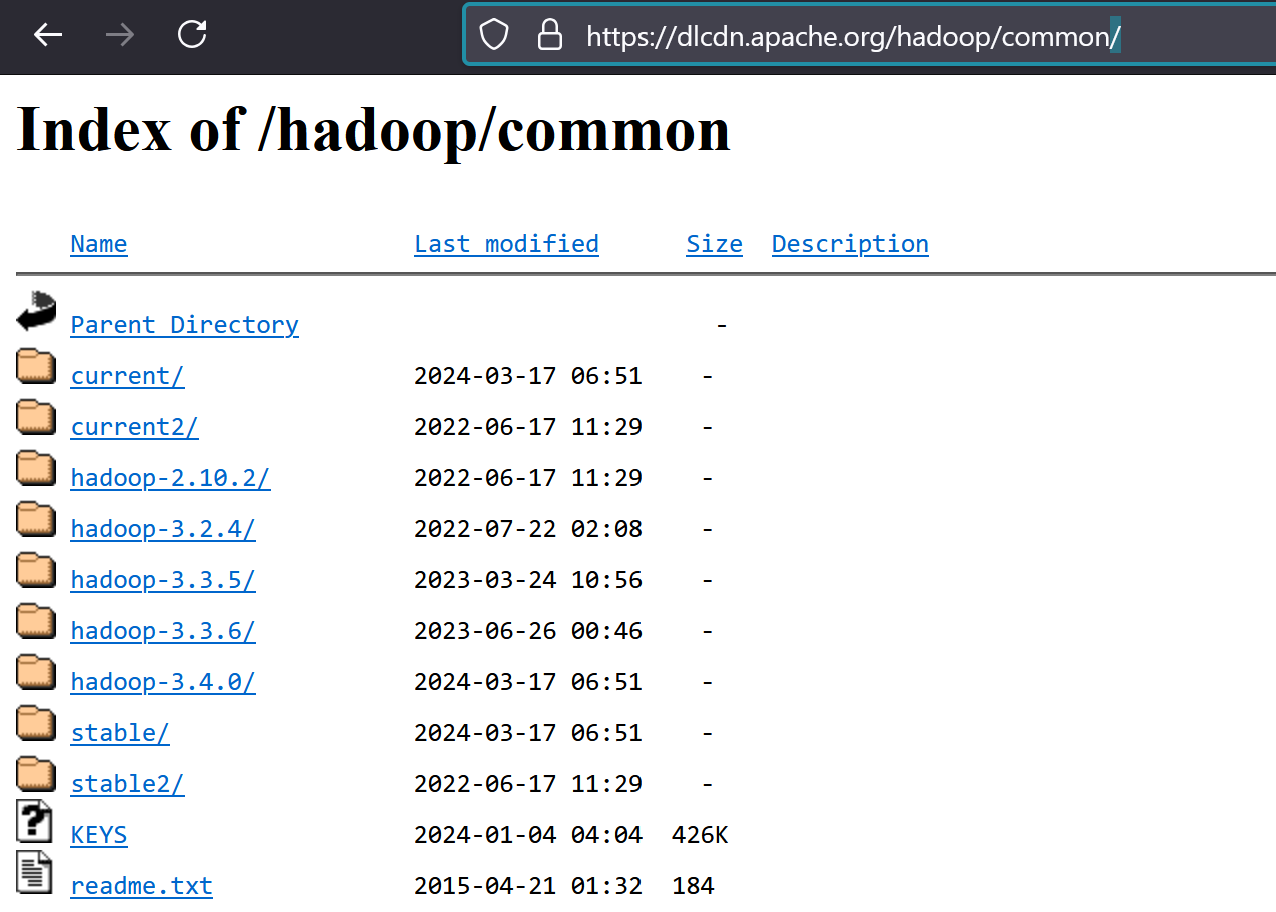
مراحل نصب Hadoop:

ابتدا از نصب بودن جاوا روی سیستم خود اطمینان حاصل میفرماییم:  


و در PATH سیستم نیز آدرس آن را اضافه میکنیم:



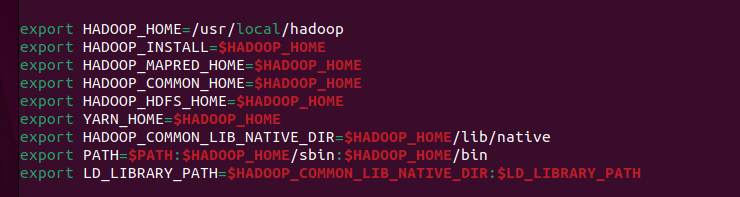
حال که سیستم آماده است از سایت اصلی آپاچی Hadoop را دانلود و از حالت فشرده خارج میکنیم:



حال نیاز است متغییر های محیطی(محیط سیستم) را برای Hadoop در bashrc تعریف کنیم و سپس تغییرات را اعمال کنیم، با استفاده از دستورات زیر:

nano ~/.bashrc

source ~/.bashrc



حال با اعمال تغییرات که تنظیمات پیش از اجرا هستند و در سایت اصلی به آن اشاره شده است، تنظیمات فرموده شده را به چهار فایل زیر اضافه میکنیم:

nano $HADOOP\_HOME/etc/hadoop/core-site.xml

nano $HADOOP\_HOME/etc/hadoop/hdfs-site.xml

nano $HADOOP\_HOME/etc/hadoop/mapred-site.xml

nano $HADOOP\_HOME/etc/hadoop/yarn-site.xml

**core-site.xml:** این فایل تنظیمات اصلی و پایه سیستم Hadoop را مشخص می‌کند. به عنوان مثال، این فایل شامل تنظیمات مربوط به سیستم فایل توزیع‌شده Hadoop (HDFS) و نحوه دسترسی به آن است.

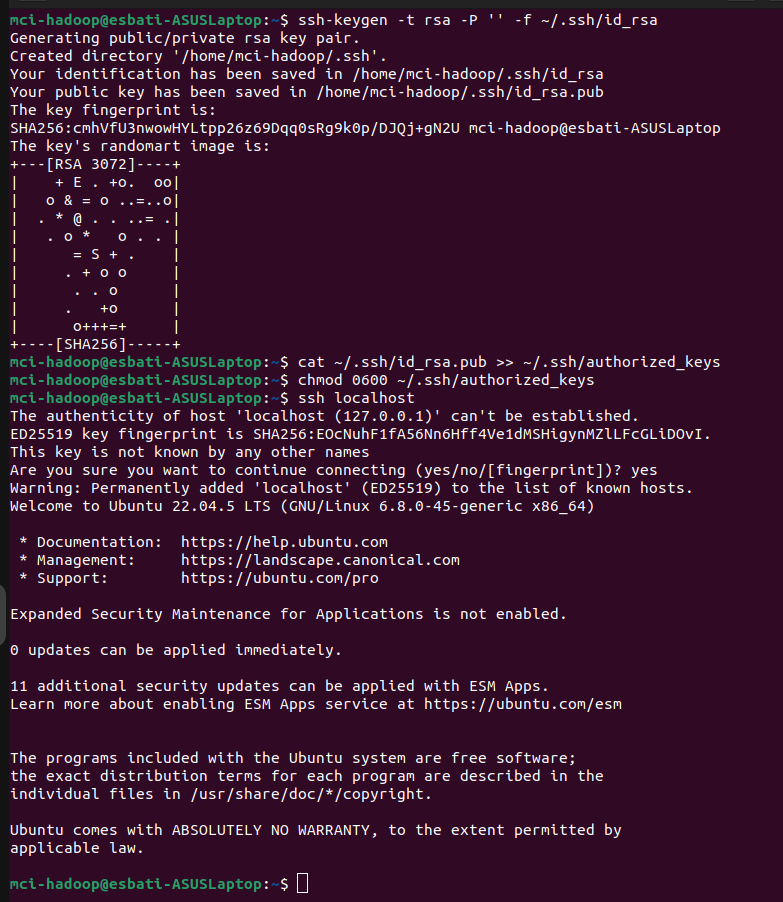
**hdfs-site.xml:** این فایل پیکربندی سیستم فایل توزیع‌شده Hadoop (HDFS) را تعیین می‌کند.   
HDFS یک سیستم فایل توزیع‌شده است که Hadoop برای ذخیره‌سازی داده‌های بزرگ استفاده می‌کند.

**mapred-site.xml:** این فایل مربوط به تنظیمات MapReduce، چارچوبی برای پردازش موازی داده‌ها در Hadoop است. در این فایل، پارامترهای مربوط به MapReduce تنظیم می‌شوند.

**yarn-site.xml:** این فایل برای تنظیماتYARN (Yet Another Resource Negotiator)، سیستم مدیریت منابع درHadoop، استفاده می‌شود. YARN مسئول مدیریت منابع در خوشه Hadoop و اجرای وظایف پردازشی در آن است.

مرحله بعد: راه‌اندازی SSH بدون رمز عبور (Passwordless SSH)

راه‌اندازی SSH بدون رمز عبور (Passwordless SSH) در بسیاری از سناریوها، به ویژه در محیط‌های توزیع‌شده مانند **Hadoop**  یا دیگر سیستم‌های توزیع‌شده، به کار می‌رود.



**چرا نیاز به SSH بدون رمز عبور داریم؟**

1. **اتوماسیون ارتباطات بین سرورها**: در محیط‌هایی مانند Hadoop، که دارای معماری توزیع‌شده است، نودهای مختلف (ماشین‌ها یا سرورها) باید به‌طور مکرر و خودکار با یکدیگر ارتباط برقرار کنند. برای مثال، DataNode ها و NameNode ها باید بتوانند بدون نیاز به وارد کردن رمز عبور از طریق SSH با یکدیگر تبادل داده کنند.
2. **اجرای دستورات از راه دور**: در Hadoop و سایر سیستم‌های توزیع‌شده، نیاز است که از طریق SSH بتوانیم از راه دور دستورات را روی نودهای دیگر اجرا کنیم. Passwordless SSH امکان می‌دهد که دستورات از طریق SSH به طور خودکار و بدون نیاز به وارد کردن رمز عبور اجرا شوند.
3. **سهولت در مدیریت خوشه‌ها (Clusters):** در زمانی که مدیریت یک خوشه شامل چندین نود است، وارد کردن مکرر رمز عبور برای هر عملیات بین نودها بسیار زمان‌بر و دست‌وپاگیر خواهد بود. SSH بدون رمز عبور این فرآیند را تسهیل می‌کند.

مرحله بعد: (اجرای فرمان hdfs namenode -format)

فرمت کردن **NameNode**  در Hadoop یکی از مراحل اولیه در راه‌اندازی سیستم فایل توزیع‌شده Hadoop (HDFS) است و برای آماده‌سازی NameNode جهت مدیریت سیستم فایل توزیع‌شده به کار می‌رود. در این مرحله، متادیتای سیستم فایل HDFS ایجاد و پیکربندی می‌شود تا HDFS بتواند کار خود را به درستی شروع کند.

**مفهوم و اهمیت فرمت کردن NameNode**

1. **NameNode چیست؟**
   * **NameNode** یک جزء مرکزی در HDFS است که وظیفه نگهداری و مدیریت **متادیتای** سیستم فایل توزیع‌شده را برعهده دارد. متادیتا شامل اطلاعاتی مانند ساختار دایرکتوری‌ها، فایل‌ها، و محل بلوک‌های داده در DataNodeها است.
   * به عبارت ساده، NameNode به نوعی "نقشه" سیستم فایل است که نشان می‌دهد فایل‌های مختلف چگونه در بلوک‌های مختلف در سراسر خوشه توزیع شده‌اند.
2. **چرا باید NameNode را فرمت کنیم؟**
   * فرمت کردن NameNode یک **مرحله اولیه** برای راه‌اندازی HDFS است. این عملیات برای آماده‌سازی NameNode و ساخت ساختارهای مورد نیاز متادیتای سیستم فایل استفاده می‌شود. بدون این مرحله، NameNode نمی‌تواند فایل‌ها را مدیریت کرده یا ارتباط صحیحی با DataNodeها برقرار کند.
   * در طی این فرایند، **دایرکتوری‌های ذخیره‌سازی متادیتا** که توسط NameNode استفاده می‌شود (مثل dfs.name.dir که در فایل hdfs-site.xml مشخص شده) فرمت می‌شوند و فایل‌های سیستمی پایه مورد نیاز HDFS در آنها ایجاد می‌شود.

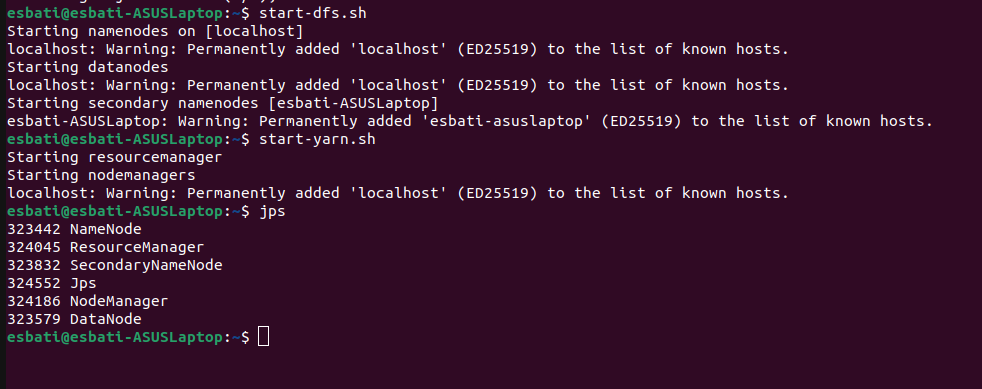
**چه زمانی این کار انجام می‌شود؟**

* **برای اولین بار پس از نصب Hadoop**: زمانی که برای اولین بار یک خوشه Hadoop راه‌اندازی می‌کنید، فرمت کردن NameNode ضروری است تا NameNode بتواند متادیتای HDFS را از ابتدا بسازد.
* **در هنگام بازسازی یا ریست سیستم فایل**: اگر قصد داشته باشید یک سیستم فایل توزیع‌شده جدید را از ابتدا ایجاد کنید یا NameNode به دلایلی خراب شده باشد و نیاز به راه‌اندازی دوباره باشد، باید آن را فرمت کنید.

و در نهایت اجرا دستورات:

start-dfs.sh

start-yarn.sh



حال برای انتقال فایل های لاگ ایجاد شده در مرحله قبل کافی است، مراحل زیر را طی کنیم:

# Create HDFS directory

hdfs dfs -mkdir /log

# Move all log files to HDFS

hdfs dfs -put log-\*.log /log

# Verify files in HDFS

hdfs dfs -ls /log

و در نهایت:

