# Advanced Linux and Command Line Tools

Md Rasheduzzaman

2025-09-20

Advanced text processing, scripting, and system administration

## Table of contents

Learn a lot from Harvard Chan Bioinformatics Core

# 1 Prerequisites

Before diving into this advanced material, you should be comfortable with:

- **Basic Linux commands**: `ls`, `cd`, `pwd`, `mkdir`, `cp`, `mv`, `rm`, `cat`, `head`, `tail`
- **File permissions**: Understanding `chmod` and `chown`
- **Text editors**: Basic usage of `nano` or `vim`
- **File system navigation**: Understanding directory structure and paths
- **Basic shell concepts**: Environment variables, command history, tab completion

If you're not familiar with these concepts, please complete the Linux Basics tutorial first.

# 2 Advanced Text Processing Tools

## 2.1 `grep` - Advanced Pattern Matching

`grep` is one of the most powerful text search tools. Let's explore its advanced features.

### 2.1.1 Basic grep Usage

```
# Search for a pattern in a file
grep "pattern" file.txt

# Search case-insensitive
grep -i "pattern" file.txt

# Search in multiple files
grep "pattern" *.txt

# Search recursively in directories
grep -r "pattern" directory/
```

### 2.1.2 Advanced grep Options

```
# Show line numbers
grep -n "pattern" file.txt

# Show context (2 lines before and after)
grep -C 2 "pattern" file.txt

# Show only filenames with matches
grep -l "pattern" *.txt

# Show only lines that DON'T match (invert)
grep -v "pattern" file.txt

# Use regular expressions
grep -E "^[0-9]+" file.txt  # Lines starting with numbers

# Count matches
grep -c "pattern" file.txt

# Show only the matching part
grep -o "pattern" file.txt
```

### 2.1.3 Regular Expressions with grep

```
# Lines starting with specific text
grep "^start" file.txt

# Lines ending with specific text
grep "end$" file.txt

# Lines containing either pattern1 OR pattern2
grep -E "pattern1|pattern2" file.txt

# Lines with exactly 3 digits
grep -E "^[0-9]{3}$" file.txt

# Lines containing word boundaries
grep -w "word" file.txt
```

## 2.2 `awk` - Pattern Scanning and Processing

`awk` is a powerful programming language for text processing. It processes files line by line.

### 2.2.1 Basic awk Syntax

```
# Print entire lines
awk '{print}' file.txt

# Print specific fields (space-separated by default)
awk '{print $1}' file.txt  # First field
awk '{print $2}' file.txt  # Second field
awk '{print $1, $3}' file.txt  # First and third fields

# Print last field
awk '{print $NF}' file.txt

# Print number of fields in each line
awk '{print NF}' file.txt
```

### 2.2.2 Advanced awk Examples

```
# Print lines with more than 3 fields
awk 'NF > 3' file.txt

# Print lines where first field equals "name"
awk '$1 == "name"' file.txt

# Print lines where second field is greater than 100
awk '$2 > 100' file.txt

# Add line numbers
awk '{print NR, $0}' file.txt

# Print specific lines (e.g., lines 5-10)
awk 'NR >= 5 && NR <= 10' file.txt

# Calculate sum of second column
awk '{sum += $2} END {print sum}' file.txt

# Print average of second column
awk '{sum += $2; count++} END {print sum/count}' file.txt
```

### 2.2.3 awk with Field Separators

```
# Use comma as field separator
awk -F',' '{print $1, $2}' file.csv

# Use multiple separators
awk -F'[,;]' '{print $1, $2}' file.txt

# Use tab as separator
awk -F'\t' '{print $1, $2}' file.tsv
```

## 2.3 `cut` - Extract Columns from Files

`cut` is simpler than `awk` for basic column extraction.

### 2.3.1 Basic cut Usage

```
# Extract first column (space-separated)
cut -d' ' -f1 file.txt

# Extract first and third columns
cut -d' ' -f1,3 file.txt

# Extract columns 1-3
cut -d' ' -f1-3 file.txt

# Use comma as delimiter
cut -d',' -f1,2 file.csv

# Use tab as delimiter
cut -d$'\t' -f1,2 file.tsv

# Extract by character positions
cut -c1-10 file.txt  # Characters 1-10
cut -c1,5,10 file.txt  # Characters 1, 5, and 10
```

## 2.4 `sed` - Stream Editor

`sed` is used for text substitution and editing.

### 2.4.1 Basic sed Usage

```
# Replace first occurrence of "old" with "new"
sed 's/old/new/' file.txt

# Replace all occurrences of "old" with "new"
sed 's/old/new/g' file.txt

# Replace only on specific lines (e.g., line 5)
sed '5s/old/new/' file.txt

# Delete lines containing "pattern"
sed '/pattern/d' file.txt
```

```
# Delete empty lines
sed '/^$/d' file.txt

# Print only lines 5-10
sed -n '5,10p' file.txt
```

### 2.4.2 Advanced sed Examples

```
# Replace multiple patterns
sed -e 's/old1/new1/g' -e 's/old2/new2/g' file.txt

# Use different delimiter (useful for paths)
sed 's|/old/path|/new/path|g' file.txt

# Case-insensitive replacement
sed 's/old/new/gi' file.txt

# In-place editing (modify file directly)
sed -i 's/old/new/g' file.txt

# Backup original file
sed -i.bak 's/old/new/g' file.txt
```

## 2.5 `sort` - Sort Lines

```
# Sort alphabetically
sort file.txt

# Sort numerically
sort -n file.txt

# Sort in reverse order
sort -r file.txt

# Sort by specific field
sort -k2 file.txt  # Sort by second field

# Sort by multiple fields
sort -k1,1 -k2,2n file.txt  # Sort by field 1, then by field 2 numerically
```

```
# Remove duplicates while sorting
sort -u file.txt

# Sort ignoring case
sort -f file.txt
```

### 2.6 `uniq` - Remove Duplicate Lines

```
# Remove consecutive duplicate lines
uniq file.txt

# Count occurrences of each line
uniq -c file.txt

# Show only unique lines
uniq -u file.txt

# Show only duplicate lines
uniq -d file.txt

# Ignore case when comparing
uniq -i file.txt
```

## 3 Advanced File Operations

### 3.1 `find` - Find Files and Directories

```
# Find files by name
find . -name "*.txt"

# Find files by type
find . -type f  # Files only
find . -type d  # Directories only

# Find files by size
find . -size +100M  # Files larger than 100MB
find . -size -1k    # Files smaller than 1KB
```

```
# Find files by modification time
find . -mtime -7    # Modified in last 7 days
find . -mtime +30   # Modified more than 30 days ago

# Find files by permissions
find . -perm 644    # Files with 644 permissions
find . -perm -u+x   # Files executable by owner

# Execute commands on found files
find . -name "*.txt" -exec rm {} \;  # Delete all .txt files
find . -name "*.log" -exec mv {} logs/ \;  # Move .log files to logs directory
```

### 3.2 `xargs` - Execute Commands on Multiple Arguments

```
# Find and delete files
find . -name "*.tmp" | xargs rm

# Find and copy files
find . -name "*.txt" | xargs cp -t backup/

# Count lines in multiple files
find . -name "*.txt" | xargs wc -l

# Search in multiple files
find . -name "*.py" | xargs grep "import"
```

## 4 Shell Scripting Basics

### 4.1 Creating and Running Scripts

```
# Create a script file
nano my_script.sh

# Make it executable
chmod +x my_script.sh

# Run the script
./my_script.sh
```

## 4.2 Basic Script Structure

```
#!/bin/bash
# This is a comment

# Set variables
NAME="Linux User"
COUNT=10

# Use variables
echo "Hello, $NAME!"
echo "Count is: $COUNT"

# Use command substitution
CURRENT_DIR=$(pwd)
echo "Current directory: $CURRENT_DIR"

# Use arithmetic
RESULT=$((COUNT * 2))
echo "Double count: $RESULT"
```

## 4.3 Control Structures

### 4.3.1 If-Else Statements

```
#!/bin/bash

if [ -f "file.txt" ]; then
    echo "File exists"
else
```

```
    echo "File does not exist"
fi

# String comparison
if [ "$1" = "hello" ]; then
    echo "You said hello"
elif [ "$1" = "goodbye" ]; then
    echo "You said goodbye"
else
    echo "You said something else"
fi
```

### 4.3.2 Loops

```
#!/bin/bash

# For loop
for i in {1..5}; do
    echo "Number: $i"
done

# For loop with files
for file in *.txt; do
    echo "Processing: $file"
done

# While loop
count=1
while [ $count -le 5 ]; do
    echo "Count: $count"
    count=$((count + 1))
done
```

## 4.4 File Manipulation Scripts

### 4.4.1 Q1: How to move files to subfolders

I have many files inside a folder. I want to move them into two sub-folders named `f1` and `f2` (I made them using `mkdir -p f1 f2`). How to do that?

This is the way:

```bash
#!/bin/bash
# move_files.sh

# Create subdirectories if they don't exist
mkdir -p f1 f2

# Counter for files
i=0

# Loop through all .fastq.gz files (sorted by version)
for file in $(ls *.fastq.gz | sort -V); do
    if [ $i -le 400 ]; then
        mv "$file" f1/
        echo "Moved $file to f1/"
    else
        mv "$file" f2/
        echo "Moved $file to f2/"
    fi
    i=$((i + 1))
done

echo "File moving completed!"
echo "Files in f1: $(ls f1/ | wc -l)"
echo "Files in f2: $(ls f2/ | wc -l)"
```

Here, I am sending `fastq.gz` files having 0-400 in their name to `f1` folder and remaining ones to `f2`. You just need to use your file naming pattern in the code block, and you are all set. For example, if you have many `.fasta` files, use `.fasta` instead of `.fastq.gz`. You got the idea, right?

**How to execute/run this file now?** Run these:

```bash
chmod +x move_files.sh
./move_files.sh
```

`chmod +x` is making the file named `move_files.sh` executable. Then we are running it using `./move_files.sh`.

# 5 High Performance Computing (HPC) Basics

## 5.1 SSH Connection to HPC Clusters

I am going to connect to Uni-Greifswald's Brain cluster.

```
ssh username@brain.uni-greifswald.de
```

**You have to use your real username and password.** Now, let's get an interactive session to the gpu compute node (it is named "vision" for uni-greifswald's gpu node, check for yours).

```
srun --pty --gres=gpu:1 --partition=vision --mem=16g -t 12:00:00 bash -i
```

So, I am taking the session for 12 hours.

## 5.2 Environment Management on HPC

### 5.2.1 Installing Conda on HPC

Let's install conda for our environment management (if you don't have already).

```
# Download the Miniconda installer for Linux
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

# Run the installer, specifying the installation path
bash Miniconda3-latest-Linux-x86_64.sh -b -p ~/miniconda_install/
```

Now, let's initialize conda:

```
# Source the conda shell script to make the 'conda' command available
source ~/miniconda_install/bin/activate

# Initialize conda for the current shell session.
conda init bash
```

Since we're using an interactive session, we won't need to manually source anything after this. The `conda init` command makes it so we can use `conda` and `conda activate` as we normally would.

**N.B. We could make some aliases for conda commands to write conda codes in shorter format, but we can do/see it later. Get used to the normal ones.**

## 5.3 Advanced Environment Setup

Let's make an environment and install our required tools there. Well, our goal is to make a system where we will use some images/characters and make short videos using them to teach Italian. We need to process photos, images, text, and sync lips (video) with audio/speech. We need MuseTalk for this. Let's configure our environment accordingly.

**Step 1 – Create conda environment**

```
# Create new environment for MuseTalk
conda create -n musetalk python=3.10
conda activate musetalk
```

**Step 2 – Install Dependencies**

```
# Install PyTorch (adapt cuda version to your cluster, here CUDA 11.8 example)
pip install torch==2.1.0 torchvision==0.16.0 torchaudio==2.1.0 --extra-index-url https://d

# HuggingFace core libs
pip install transformers accelerate diffusers safetensors

# MuseTalk repo (clone)
git clone https://github.com/TMElyralab/MuseTalk.git
cd MuseTalk

# Install requirements
pip install -r requirements.txt

# Install whisper encoder
pip install --editable ./musetalk/whisper

# Extra: ffmpeg for video processing
conda install -c conda-forge ffmpeg -y
```

**Step 3 – Download MuseTalk Models**
MuseTalk Hugging Face repo: https://huggingface.co/TMElyralab/MuseTalk We need: - musetalk.pth (main model) - gfpgan (optional face enhancer)

Run inside MuseTalk:

```
mkdir checkpoints
cd checkpoints
```

```
# Download core model
wget https://huggingface.co/TMElyralab/MuseTalk/resolve/main/musetalk.pth

# Optional face enhancer
git clone https://github.com/TencentARC/GFPGAN.git
cd ..
```

## 5.4 CUDA Environment Configuration

### 5.4.1 Better way:

```
export CUDA_HOME=/usr/local/cuda-11.7
export PATH=$CUDA_HOME/bin:$PATH
export LD_LIBRARY_PATH=$CUDA_HOME/lib64:$LD_LIBRARY_PATH
```

In my HPC Cluster, it is cuda 11.7. I am configuring for that. We could add these lines in my `~/.bashrc` file as well. We might see it later.

Let's load our cuda module first to get things done smoothly.

```
module load cuda/11.7
```

Now, make the `yml` file to make the environment with all the tools required.

```
name: musetalk3
channels:
  - pytorch
  - nvidia
  - defaults
dependencies:
  - python=3.10
  - pip
  - ffmpeg
  - pip:
      # PyTorch + CUDA 11.8 compatible with 11.7 system
      - torch==2.1.0+cu118
      - torchvision==0.16.0+cu118
      - torchaudio==2.1.0+cu118
      - --extra-index-url https://download.pytorch.org/whl/cu118

      # OpenMMLab dependencies
```

```
        - mmcv==2.0.1 -f https://download.openmmlab.com/mmcv/dist/cu118/torch2.1.0/index.htm
        - mmdet==3.1.0
        - gradio
        - opencv-python
        - numpy
        - scipy
        - matplotlib
        - tqdm
        - pyyaml
        - pillow
        - soundfile
        - librosa
        - moviepy
        - imageio
```

Save this as `musetalk3.yml` and run:

```
conda env create -f musetalk3.yml
conda activate musetalk3
```

Now, we have our environment ready to use. Let's use it.

```
conda activate musetalk3
```

Now, install museetalk repo dependencies.

```
git clone https://github.com/TMElyralab/MuseTalk.git
cd MuseTalk
pip install -r requirements.txt
```

Let's get the faces I want to use. I uploaded them to my Google Drive.

```
pip install gdown

# Get shareable link from Google Drive and copy file id
gdown https://drive.google.com/uc?id=FILE_ID -O data/faces/alice.jpg
gdown https://drive.google.com/uc?id=FILE_ID -O data/faces/bob.jpg
```

Modify the google drive links for the images and their destination name as you like it.

### 5.4.2 Real work

Now, let's follow the author's guideline.

```
pip install --no-cache-dir -U openmim
mim install mmengine
mim install "mmcv==2.0.1"
mim install "mmdet==3.1.0"
mim install "mmpose==1.1.0"
```

Let's download the model's weight.

```
sh ./download_weights.sh
```

```
# Check ffmpeg installation
ffmpeg -version
```

The conversation

```
pip install TTS

# Example: Alice speaking Italian
tts --text "Buon giorno, Bob! Come stai?" \
    --model_name tts_models/it/mai_female/vits \
    --out_path alice_buongiorno.wav

# Example: Bob replying
tts --text "Buon giorno, Alice! Va bene, grazie mille! Chi è questo?" \
    --model_name tts_models/it/mai_male/vits \
    --out_path bob_risposta.wav
```