DAT151
Database and Unix System Management

Spring 2025

Assignment 1

Obligatory assignment. Deadline: ***Sunday 16.02.2025***

The report should include all necessary commands to complete the tasks, printout from the system, explanation of what is done, the result and explanation of the result.

Remember to include the names of the group members on the front page of the report. The report can be in English or Norwegian. The report should be handed in via Canvas.

The assignment should be accomplished in groups of two or three students. Signing up for a group will be closed on Thursday 13.02.

You must select a group "Lab1 N" when delivering report, also if you are the only member of the group.

***Observe**: Before you modify an existing configuration file, save a copy of the original somewhere on the computer, e.g. below a folder "/root/orig/".*

Lecturers:
Bjarte Wang-Kileng, Bjarte.Kileng@hvl.no, room D409
Haakon André Reme-Ness Haakon.Andre.Reme-Ness@hvl.no

Last changed, 29/01/25 (BK)

Part 1 – The Filesystem

Task 1: Filesystem basics

1. Describe some of the key differences between the Linux/UNIX filesystem and Windows. How are files and folders organised in Linux/UNIX?
2. What is a "mount point"?

   Create a 10GiB partition on the free part of the disk. You can e.g. use the command **parted** to create a new partition.

   If the free part is contained in a mounted partition, un-mount the partition using the **umount** command and comment out/remove the reference to the extra 10GiB partition from the file "/etc/fstab" file. Then drop the partition using e.g. the **parted** program.

Create a new folder below "/opt". Then use the **mount** command and mount the 10GiB partition on the new directory. Explain the outcome.

For help on the use of the **parted** program, see the appendix.

3. In most UNIX systems, there are seven types of files defined. What are these? Use the **file** command to display the information of a few files. In which of the seven categories do these files belong?

   Hints:
   - Remember that many things can be considered as "regular files", such as texts and executables.
   - The command **ls -l** can display the exact type of all files in a directory. Look at the very first letter of each line. If there is no letter (instead only a "-"), then it's a regular file.

4. From a user perspective, hard links and symbolic links allow the same file to exist in more than one place. Find an empty directory and create a new file with some text in it. Create a hard link for this file in the same directory using the command **ln**. Now, create a symbolic link for the file using **ln -s**. Then delete the original file, leaving only the links.

   a) Check if the hard link still works. Why/why not?

   b) Check if the soft link still works. Why/why not?

Task 2: File attributes and permissions

1. In the traditional Linux/UNIX filesystem model, every file comes bundled with a set of 16 bits. What are these bits, and why are they needed? Explain.

2. The nine permission bits are often represented using octal numbers. Explain what these are, and how the permissions bits 110 100 101 can be represented using octals. What permissions does a file with the octal value 745 have?

3. **chmod** is a tool that allows users to change the permission bits of a file, either by providing octal values, or by a mnemonic syntax where you can combine a set of targets (**u**, **g**, **o** for user, group and others, or **a** for all three) combined with an operator (**+** , **-** , **=** for add, remove or set) before providing the wanted permissions. For instance, to add read and write permissions to the group and all others for a file, the command **chmod go+rw** can be used.

   a) Create a new file and use **chmod** with the mnemonic syntax to add read and write permissions to the file owner, read and execute permissions for the group, and read and execute permissions for everyone else.

   b) Create a second file with the same permissions, but now use **chmod** with octal numbers to set the permissions.

   c) Can you think of any reasons why octal numbers can be preferred over the mnemonic syntax?

Part 2 – Software installation and management

Task 3: Package management systems and lower level package management

1. What is a package management system, and why do we need one? What is the package format used by the package management system on your machine?
2. Package management can often be considered to have two layers, where lower level package management contains tools used to install, uninstall and query packages.

   a) Use the package management tool available on your system to list all installed packages. How many are installed on your system? Select one package and use the tool again to display all its dependencies.

   Hints:

   b) Use **wget** to download the package for the application called **openvpn**. Use an official AlmaLinux repository.

   Tip: Check the URLs for AlmaLinux under /etc/yum.repos.d

   c) Try to install this package using your system's packaging tool. What happens? Why?

Task 4: High-level package management

High-level package management systems aim to simplify the process of installing, updating and maintaining packages. The two most common of these systems are **apt** (the Advanced Package Tool) and **dnf** (Dandified YUM).

1. Explain how these systems can locate and install software. What is the high-level package management system on your machine?
2. What is a software repository? If you are using *dnf* as your high-level package management system, add the **epel** software repository to your system. If you are using something else try a different repository of choice.

   Hints:
   - More information available at http://fedoraproject.org/wiki/EPEL
   - Use **dnf** and intall the "epel-release" package, that adds the **epel** software.
3. Try using **dnf** to install the **openvpn** package. Does this differ in any way from task 3? If it installs, remove the package afterwards.

   Hints:
   - The **dnf** have command named **install, remove a**nd **history**
   - The history sub command *undo* can be very useful.
   - Simply type **dnf** in your terminal to see a list of all available commands.
   - More information is available through the man system.
4. Use **dnf** to update the packages on your system.

Part 3 – Systemd

Task 1 Systemd

1. The **ctrl-alt-del.target** is started whenever *Control+Alt+Del* is pressed on the console. This target will be symlinked to another target.
   a) What is the current target that **ctrl-alt-del.target** points to, and what are the consequences?
   b) Modify the **ctrl-alt-del.target** to halt the computer when *Control+Alt+Del* is pressed.
      - Do not modify the content of the file pointed to by **ctrl-alt-del.target**. Rather, modify the target of the symlink.
2. Set the systemd **default.target** to **multi-user.target**, then restart the computer.
   - Modify the target of the symlink using the systemctl command.

   What is the consequences of the **multi-user.target**. Obs: Write down the current value of **default.target** before modifying its value.

   When in the **multi-user.target,** use the **systemctl isolate** command and start the graphical target.
3. Reboot the computer, and boot into the **emergency.target**. Mount the root file system (/) rw, and create a file in the **/root** directory. Then restart the computer.
4. Only if the previous task was solved, modify the */boot* stanza in the */etc/fstab* file and use a wrong value. **Obs.:** Take a copy of the *fstab* file before modifying it.

   Reboot the computer. When the mount fails, the kernel will start the emergency target. Use the emergency shell, and correct the */etc/fstab* file, then restart the computer.
5. Create a directory *ram* below your home directory. Create a systemd mount unit that mounts a ramdisk on the *ram* directory.

   In this task:
   - **Do not** modify the *fstab* file.
   - **Do not** use the *mount* command.

   Mounting the ramdisk should be done using "*systemctl start*".

Appendix: Create partitions

Several tools let us work with disk partitions, e.g. the **parted** program. Below are some examples on how to use **parted**.

In all examples, the disk is assumed to be "/dev/sda".

List free space on disk

The command below will display the free disk areas in addition to the disk partitions. The unit of measure is sectors, where a sector is 512 bytes.

```
parted -s /dev/sda unit s print free
```

The **parted** program can return the result as JSON using the switch “-j”, and the **jq** program is a command line JSON processor the can read JSON from STDIN. Together they make a powerful solution for querying for disk properties.

The command below will return the first sector of the last free disk area:

```
parted -s /dev/sda unit s print free -j | \
    jq 'last(.disk.partitions[]|select(.type|contains("free"))).start'
```

Create partition

The **parted** subcommand **mkpart** can be used to create partitions.

For performance reasons, partitions are usually aligned on 1MiB boundaries, or 2048 sectors. This is not a strict requirement, but some tools will complain if the disk is not properly aligned.

Assume that the larger free disk area starts at sector 457574312. The next 1MiB boundary is then:

```
echo $(( (457574312+2047)/2048 )) # prints 223425
```

This gives 223425MiB, or as sectors:

```
echo $(( 223425 * 2048 )) # prints 457574400
```

This gives the value 457574400. That is, the next 1MiB boundary will start at sector 457574400.

A 10GiB partition as sectors is:

```
echo $(( 10*1024*1024*2)) # prints 20971520
```

That is, a 10GiB partition will count 20971520 sectors.

The new 10GiB partition can therefore end at sector:

```
echo $(( 457574400 + 20971520)) # prints 478545920
```

The result is 478545920, but to reduce the amount of unused space, the following sector, sector 478545921 should be a 1MiB boundary:

```
echo $(( (478545921 + 2047)/2048 )) # prints 233666
```

The next 1MiB boundary is 233666MiB, or as sectors:

```
echo $(( 233666 * 2048 )) # prints 478547968
```

That is, the next 1MiB boundary is at sector 478547968. The partition can therefore be filled to the partition before, i.e. sector 478547967:

```
parted -s /dev/sda '"almaone partition"' xfs 457574400s 478547967s
```

Use a label containing *almatwo* for the second system.

The 1MiB alignment can also be achieved using[1]:

```
parted -s /dev/sda '"almaone partition"' xfs 233666MB \
    $((233666+10*1024))MB
```

If the partition is to fill all remaining disk space with the end position respecting the 1MiB boundary rule, the end position can be given as 100%:

```
parted -s /dev/sda '"almaone partition"' xfs 457574400s 100%
```

1 The parted program can work also with e.g. the units “MiB” and “MB”. With unit “MB”, **parted** will compute sensible values to use. The unit “s” for a sector of 512 bytes is the smallest addressable disk unit, although modern disks internally often uses a sector size of 4KiB.

Create an xfs file system on partition

Before the newly created partition can be mounted, a file system must be created on the partition. The program to use is **mkfs**.

First, locate the partition number of the newly created partition, e.g. using the **parted** program. If the number is e.g. 6, the device will be “/dev/sda6”. The command to create an XFS file system on “/dev/sda6” will be:

```
mkfs -t xfs /dev/sda6
```