RHCSA EX200 Bonus Practice Book

Advanced Lab Tasks, Podman, ACL, Text Processing & Full Mock Exams

Preface

The Red Hat Certified System Administrator (RHCSA) EX200 exam is one of the most respected and rigorous performance-based certifications in the Linux world. It doesn't ask you to recall facts from memory or select answers from a list—it demands that you *perform*. You sit in front of a live system, and you either solve the problems or you don't. There is no partial credit for theory you understand but cannot execute.

This book exists because understanding is not enough. You need to be fast, accurate, and calm under pressure.

Purpose and Scope

The *RHCSA EX200 Bonus Practice Book* is designed as a focused companion for candidates who have already studied the core RHCSA curriculum and are ready to sharpen their skills through intensive, hands-on practice. This is not a textbook that introduces concepts from scratch. It is a training ground—a collection of targeted lab exercises, timed drills, troubleshooting scenarios, and full-length mock exams built to push you beyond your comfort zone and into exam readiness.

Every task in this book is crafted to mirror the pressure, ambiguity, and complexity of the real RHCSA EX200 environment. If you can complete these exercises confidently, you will walk into the exam with a decisive advantage.

What This Book Covers

The sixteen chapters are organized around the RHCSA domains that candidates most frequently struggle with or underestimate:

- **Chapters 1–4** focus on essential file operations, text processing, archive manipulation, advanced permissions, and Access Control Lists (ACLs)—skills that form the foundation of daily system administration and appear consistently on the RHCSA exam.
- **Chapters 5–7** tackle SELinux troubleshooting, disk partitioning, filesystem creation, and LVM configuration through timed drills that build both competence and speed.
- **Chapters 8–10** immerse you in systemd emergency recovery, network reconfiguration, and firewalld management—scenarios where a single misconfiguration can lock you out of your own system.
- **Chapters 11–12** provide dedicated coverage of *Podman*, which has become an increasingly important component of the modern RHCSA exam. You will practice container management, persistent storage, and systemd integration, as well as troubleshoot broken container configurations.
- **Chapters 13–14** drill cron scheduling and Bash mini-automation tasks, reinforcing the automation skills that the RHCSA exam expects you to demonstrate.
- **Chapters 15–16** deliver two complete mock exams—one standard and one advanced—designed to simulate the full RHCSA experience from start to finish.

How to Use This Book

Work through each chapter at the terminal. Read the scenario, attempt the task *before* looking at the solution, and time yourself. Repeat exercises until the commands flow from muscle memory. The advanced mock exam in Chapter 16 is intentionally harder than the real RHCSA exam—if you can pass it, the actual test will feel manageable.

Who This Book Is For

This book is for RHCSA candidates who want more practice than a single study guide provides. Whether you are attempting the EX200 for the first time or retaking it after a near miss, these labs will expose your weak points and give you the repetitions needed to eliminate them.

Acknowledgments

This book would not exist without the vibrant community of Linux professionals and RHCSA candidates who share their experiences, frustrations, and insights. I am grateful to the open-source community and to Red Hat for maintaining a certification that genuinely measures real-world skill. Special thanks to every reader who chose to invest their time in deliberate practice—your commitment to excellence is what makes this work meaningful.

The terminal is waiting. Let's get to work.

Miles Everhart

Table of Contents

Chapter	Title	Page
1	Advanced Text Processing Tasks	6
2	File and Archive Manipulation Drills	21
3	Advanced Permission Scenarios	34
4	ACL Intensive Lab	50
5	SELinux Debug & Repair	64
6	Partition & Filesystem Timed Drills	76
7	LVM Rapid Configuration Challenges	91
8	systemd Emergency Scenarios	103
9	Network Reconfiguration Under Pressure	117
10	firewalld Advanced Tasks	129
11	Podman Essentials for RHCSA	142
12	Podman Troubleshooting Labs	154
13	Cron & Automation Drills	165
14	Bash Mini-Automation Tasks	182
15	Full Mock Exam #1	199
16	Full Mock Exam #2 (Advanced)	218

Chapter 1: Advanced Text Processing Tasks

Text processing is one of the most critical skills you will need to master for the Red Hat Certified System Administrator (RHCSA) EX200 exam. While many candidates focus heavily on storage management, user administration, and service configuration, the ability to efficiently search, filter, extract, transform, and manipulate text from files and command output is equally important. In a real-world enterprise Linux environment, system administrators spend a significant portion of their time reading log files, parsing configuration files, extracting specific data from command output, and automating repetitive text manipulation tasks. The RHCSA exam expects you to be proficient with a range of text processing utilities, and this chapter will take you well beyond the basics.

Throughout this chapter, you will work with tools such as `grep`, `sed`, `awk`, `cut`, `sort`, `uniq`, `tr`, `wc`, `tee`, `head`, `tail`, and various combinations of these commands using pipes. Each tool serves a specific purpose, and understanding when and how to combine them is what separates a competent administrator from a novice. We will begin with foundational concepts, build up to advanced usage patterns, and conclude with hands-on lab exercises that mirror the complexity and pressure of the actual RHCSA exam environment.

Understanding the Linux Text Processing Philosophy

Before diving into individual commands, it is essential to understand the Unix and Linux philosophy that underpins text processing. Linux was designed around the concept of small, specialized tools that do one thing well. These tools communicate with each other through standard input (stdin), standard output (stdout), and standard error (stderr). The pipe operator (`|`) allows you to chain these tools together, creating powerful data processing pipelines from simple building blocks.

For example, consider a scenario where you need to find all users on a system who use the `/bin/bash` shell, sort them alphabetically, and count how many there are. Rather than writing a complex program, you would chain together several simple commands:

```
grep '/bin/bash' /etc/passwd | cut -d: -f1 | sort | wc -l
```

This single pipeline uses four commands, each performing one specific task. The `grep` command filters lines containing `/bin/bash`, the `cut` command extracts the username field, the `sort` command arranges them alphabetically, and the `wc -l` command counts the total number of lines. This approach is fundamental to everything you will do in this chapter.

Working with grep for Pattern Searching

The `grep` command is arguably the most frequently used text processing tool on any Linux system. Its name stands for "Global Regular Expression Print," and its primary purpose is to search for patterns within files or input streams and display matching lines.

The basic syntax of `grep` is straightforward:

```
grep [options] 'pattern' filename
```

However, the power of `grep` lies in its options and its support for regular expressions. The following table provides a comprehensive reference for the most important `grep` options you should know for the RHCSA exam:

Option	**Description**	**Example**
`-i`	Performs a case-insensitive search	`grep -i 'error' /var/log/messages`
`-v`	Inverts the match, showing lines that do NOT match	`grep -v '^#' /etc/ssh/sshd_config`
`-r` or `-R`	Searches recursively through directories	`grep -r 'PermitRoot' /etc/ssh/`
`-l`	Displays only the names of files containing matches	`grep -rl 'SELINUX' /etc/`
`-n`	Displays line numbers alongside matching lines	`grep -n 'failed' /var/log/secure`
`-c`	Counts the number of matching lines	`grep -c 'root' /etc/passwd`
`-w`	Matches only whole words	`grep -w 'root' /etc/passwd`
`-E`	Enables extended regular expressions (equivalent to `egrep`)	`grep -E 'error|warning|critical' /var/log/messages`
`-o`	Displays only the matching portion of the line	`grep -o '[0-9]\+\.[0-9]\+\.[0-9]\+\.[0-9]\+' /var/log/secure`

`-A n`	Shows n lines after each match	`grep -A 3 'error' /var/log/messages`
`-B n`	Shows n lines before each match	`grep -B 2 'error' /var/log/messages`
`-C n`	Shows n lines before and after each match (context)	`grep -C 2 'error' /var/log/messages`

Note: The difference between basic regular expressions (used by default with `grep`) and extended regular expressions (used with `grep -E` or `egrep`) is important. With basic regular expressions, metacharacters such as `+`, `?`, `{`, `}`, `(`, `)`, and `|` must be escaped with a backslash to be treated as special. With extended regular expressions, these characters have special meaning by default.

Let us look at some practical examples that are directly relevant to RHCSA tasks:

Finding all uncommented, non-empty lines in a configuration file:

```
grep -v '^#' /etc/ssh/sshd_config | grep -v '^$'
```

This is an incredibly useful pattern. The first `grep -v '^#'` removes all comment lines (lines starting with `#`), and the second `grep -v '^$'` removes all empty lines. What remains are only the active configuration directives.

Searching for failed login attempts in the security log:

```
grep -i 'failed' /var/log/secure | tail -20
```

Finding all configuration files under `/etc` that reference a specific IP address:

```
grep -rl '192.168.1.100' /etc/
```

The Power of sed for Stream Editing

The `sed` command, short for "stream editor," is a non-interactive text editor that processes text line by line. It is one of the most powerful tools available for text transformation, and the RHCSA exam frequently tests your ability to use it for search-and-replace operations, line deletion, and text insertion.

The basic syntax of `sed` is:

```
sed [options] 'command' filename
```

The most common `sed` operation is substitution, which follows this pattern:

```
sed 's/old_pattern/new_pattern/flags' filename
```

Here is a comprehensive reference table for `sed` commands and flags:

Command/Flag	Description	Example
`s/old/new/`	Substitutes the first occurrence of "old" with "new" on each line	`sed 's/http/https/' config.txt`
`s/old/new/g`	Substitutes all occurrences of "old" with "new" on each line (global)	`sed 's/old/new/g' file.txt`
`s/old/new/gi`	Global substitution, case-insensitive	`sed 's/error/WARNING/gi' log.txt`
`-i`	Edits the file in place (modifies the original file)	`sed -i 's/old/new/g' file.txt`
`-i.bak`	Edits in place but creates a backup with the specified extension	`sed -i.bak 's/old/new/g' file.txt`
`d`	Deletes lines matching a pattern	`sed '/^#/d' config.txt`

`p`	Prints lines matching a pattern (usually used with `-n`)	`sed -n '/error/p' log.txt`
`-n`	Suppresses automatic printing of pattern space	`sed -n '5,10p' file.txt`
`a\text`	Appends text after a matching line	`sed '/pattern/a\new line' file.txt`
`i\text`	Inserts text before a matching line	`sed '/pattern/i\new line' file.txt`
`c\text`	Replaces an entire matching line with new text	`sed '/old line/c\new line' file.txt`

Note: The `-i` flag is critically important for the RHCSA exam. Without it, `sed` only displays the modified output to the terminal without changing the original file. When the exam asks you to modify a configuration file, you must use `-i` to make the changes permanent.

Practical examples relevant to RHCSA tasks:

Changing the default SSH port in the SSH configuration file:

```
sed -i 's/^#Port 22/Port 2222/' /etc/ssh/sshd_config
```

Removing all comment lines and empty lines from a configuration file and saving the result to a new file:

```
sed '/^#/d;/^$/d' /etc/ssh/sshd_config > /tmp/sshd_clean.conf
```

Replacing a hostname throughout a configuration file:

```
sed -i 's/oldserver\.example\.com/newserver.example.com/g' /etc/
hosts
```

Note: When your search pattern contains special characters such as dots, forward slashes, or asterisks, you must escape them with a backslash. Alternatively, you can use a different delimiter. For example, when working with file paths, using `|` or `#` as the delimiter is often cleaner:

```
sed -i 's|/var/log/old|/var/log/new|g' /etc/rsyslog.conf
```

Text Extraction with cut, awk, and Related Tools

While `grep` finds lines and `sed` transforms them, `cut` and `awk` are primarily used for extracting specific fields or columns from structured text data.

The `cut` command is simple and efficient for extracting fields from delimited text:

```
cut -d: -f1 /etc/passwd
```

This command uses the colon (`:`) as a delimiter (`-d:`) and extracts the first field (`-f1`), which is the username in the `/etc/passwd` file.

Option	Description	Example
`-d`	Specifies the field delimiter	`cut -d: -f1 /etc/passwd`
`-f`	Specifies which fields to extract	`cut -d: -f1,3 /etc/passwd`
`-c`	Extracts specific character positions	`cut -c1-8 /etc/passwd`
`-f1-3`	Extracts a range of fields	`cut -d: -f1-3 /etc/passwd`
`--complement`	Extracts everything except the specified fields	`cut -d: -f2 --complement /etc/passwd`

The `awk` command is far more powerful than `cut` and is essentially a complete programming language for text processing. For the RHCSA exam, you need to understand its basic usage for field extraction and conditional processing.

The basic `awk` syntax is:

```
awk 'pattern { action }' filename
```

By default, `awk` uses whitespace (spaces and tabs) as the field delimiter, which makes it ideal for processing command output:

```
df -h | awk '{print $1, $5}'
```

This command runs `df -h` to show disk usage and then uses `awk` to print only the first field (filesystem name) and the fifth field (usage percentage).

Here are more practical `awk` examples for RHCSA preparation:

Extracting usernames and their shells from `/etc/passwd`:

```
awk -F: '{print $1, $7}' /etc/passwd
```

The `-F:` option tells `awk` to use the colon as the field separator, just like `cut -d:`.

Finding users with UID greater than or equal to 1000 (regular users):

```
awk -F: '$3 >= 1000 {print $1, $3}' /etc/passwd
```

This demonstrates `awk`'s ability to perform conditional processing, something that `cut` cannot do.

Displaying processes consuming more than 1% CPU:

```
ps aux | awk '$3 > 1.0 {print $1, $2, $3, $11}'
```

Sorting, Counting, and Deduplication

The `sort`, `uniq`, and `wc` commands are essential companions to the tools discussed above. They are frequently used at the end of a pipeline to organize and summarize results.

Command	**Description**	**Example**
`sort`	Sorts lines alphabetically by default	`sort /etc/passwd`
`sort -n`	Sorts numerically	`du -sh /var/log/* | sort -n`
`sort -r`	Sorts in reverse order	`sort -r /etc/passwd`
`sort -t: -k3 -n`	Sorts by the third field using colon as delimiter, numerically	`sort -t: -k3 -n /etc/passwd`
`sort -u`	Sorts and removes duplicate lines	`sort -u names.txt`
`uniq`	Removes adjacent duplicate lines (input must be sorted)	`sort access.log | uniq`
`uniq -c`	Counts occurrences of each unique line	`sort access.log | uniq -c`
`uniq -d`	Shows only duplicate lines	`sort file.txt | uniq -d`
`wc -l`	Counts lines	`wc -l /etc/passwd`
`wc -w`	Counts words	`wc -w document.txt`
`wc -c`	Counts bytes	`wc -c file.txt`

A practical example combining several of these tools to find the top 10 IP addresses in an Apache access log:

```
awk '{print $1}' /var/log/httpd/access_log | sort | uniq -c |
sort -rn | head -10
```

This pipeline extracts the first field (IP address) from each log line, sorts them so identical IPs are adjacent, counts occurrences of each unique IP, sorts the counts in reverse numerical order, and displays the top 10.

Character Translation and Transformation with tr

The `tr` command translates or deletes characters. Unlike most other text processing commands, `tr` does not accept a filename as an argument. It only reads from standard input, so you must use it with pipes or input redirection.

```
echo "Hello World" | tr 'a-z' 'A-Z'
```

This converts all lowercase letters to uppercase. Common `tr` operations include:

Operation	Description	Example
`tr 'a-z' 'A-Z'`	Converts lowercase to uppercase	`echo "hello" | tr 'a-z' 'A-Z'`
`tr 'A-Z' 'a-z'`	Converts uppercase to lowercase	`echo "HELLO" | tr 'A-Z' 'a-z'`
`tr -d 'character'`	Deletes all occurrences of a character	`echo "a1b2c3" | tr -d '0-9'`
`tr -s ' '`	Squeezes repeated characters into a single occurrence	`echo "hello    world" | tr -s ' '`
`tr ':' '\t'`	Replaces colons with tabs	`cat /etc/passwd | tr ':' '\t'`

Using tee for Dual Output

The `tee` command reads from standard input and writes to both standard output and one or more files simultaneously. This is useful when you want to save the output of a command while also viewing it on the screen or passing it to another command:

```
grep 'error' /var/log/messages | tee /tmp/errors.txt | wc -l
```

This command finds all error lines in the messages log, saves them to `/tmp/errors.txt`, and also counts them. The `-a` option appends to the file instead of overwriting it:

```
grep 'warning' /var/log/messages | tee -a /tmp/errors.txt
```

Advanced Lab Exercises

The following exercises are designed to simulate the type of text processing tasks you might encounter on the RHCSA EX200 exam. Work through each one carefully, and try to complete them without referring back to the explanations above.

Exercise 1: Extracting System Information

Create a script or a single command pipeline that extracts all usernames from `/etc/passwd` that have a home directory under `/home/`, sorts them alphabetically, and saves the output to `/tmp/home_users.txt`.

```
grep '/home/' /etc/passwd | cut -d: -f1 | sort > /tmp/
home_users.txt
```

Exercise 2: Configuration File Cleanup

Using `sed`, create a clean version of `/etc/ssh/sshd_config` that contains no comment lines (lines starting with `#`) and no empty lines. Save the output to `/tmp/sshd_active.conf`.

```
sed '/^#/d;/^$/d;/^[[:space:]]*$/d' /etc/ssh/sshd_config > /tmp/
sshd_active.conf
```

Note: The pattern `/^[[:space:]]*$/d` also removes lines that contain only whitespace characters, which is a common edge case that the simpler `/^$/d` would miss.

Exercise 3: Log Analysis

Write a command that finds all unique dates on which "Failed password" attempts appear in `/var/log/secure`, counts how many attempts occurred on each date, and displays the results sorted by count in descending order.

```
grep 'Failed password' /var/log/secure | awk '{print $1, $2}' |
sort | uniq -c | sort -rn
```

Exercise 4: In-Place File Modification

The file `/etc/issue` displays a message before the login prompt. Using `sed`, replace the current contents with "Authorized access only." Make sure the change persists.

```
echo "Authorized access only." | sudo tee /etc/issue
```

Alternatively, using `sed` to replace all content:

```
sudo sed -i 'c\Authorized access only.' /etc/issue
```

Exercise 5: Complex Pipeline

Find all installed packages on the system whose names start with "python", extract only the package names (without version numbers), sort them, remove duplicates, count the total, and display both the list and the count.

```
rpm -qa | grep '^python' | sed 's/-[0-9].*//' | sort -u | tee /
tmp/python_packages.txt | wc -l
cat /tmp/python_packages.txt
```

Exercise 6: Field Extraction and Conditional Processing

Using awk, display all users from /etc/passwd whose UID is between 1000 and 5000 (inclusive), showing only the username, UID, and default shell, separated by tabs.

```
awk -F: '$3 >= 1000 && $3 <= 5000 {print $1"\t"$3"\t"$7}' /etc/
passwd
```

Exercise 7: Search and Replace with Special Characters

A configuration file at /tmp/app.conf contains the line basedir=/opt/old/application. Using sed, change this to basedir=/opt/new/application in place.

First, create the test file:

```
echo "basedir=/opt/old/application" > /tmp/app.conf
```

Then perform the replacement using an alternate delimiter:

```
sed -i 's|basedir=/opt/old/application|basedir=/opt/new/
application|' /tmp/app.conf
```

Exercise 8: Combining Multiple Tools

Create a report that shows the top 5 largest directories under /var, displaying only the size and directory name, sorted from largest to smallest.

```
du -sh /var/*/ 2>/dev/null | sort -rh | head -5
```

Note: The 2>/dev/null redirects standard error to discard any permission denied messages, which is a common practice in RHCSA exam scenarios.

Exercise 9: Working with Regular Expressions

Find all lines in /etc/passwd where the username contains exactly four characters.

```
grep -E '^[^:]{4}:' /etc/passwd
```

This uses an extended regular expression that matches exactly four non-colon characters at the beginning of the line, followed by a colon (the field delimiter).

Exercise 10: Multi-step Text Transformation

Extract the list of all groups from `/etc/group`, remove any system groups (GID less than 1000), display only the group name and GID separated by a tab, and save the result to `/tmp/user_groups.txt`.

```
awk -F: '$3 >= 1000 {print $1"\t"$3}' /etc/group | sort -t$'\t'
-k2 -n > /tmp/user_groups.txt
```

Summary of Key Points for the RHCSA Exam

When preparing for the RHCSA EX200 exam, remember these critical text processing principles. First, always verify your commands before using the `-i` flag with `sed`. Run the command without `-i` first to see what changes would be made, and only add `-i` when you are confident the output is correct. Second, understand that `uniq` requires sorted input to function correctly. If you use `uniq` without first sorting the data, it will only remove adjacent duplicates, potentially missing duplicates that are separated by other lines. Third, practice building pipelines incrementally. Start with the first command, verify its output, then add the next command in the pipeline, and continue until you have the complete solution. Fourth, remember that `awk` is your most versatile tool for field extraction with conditions, while `cut` is faster and simpler for straightforward field extraction from delimited files. Fifth, always consider edge cases such as lines with only whitespace, mixed case text, and special characters in file paths when constructing your patterns.

Mastering these text processing tools will not only help you pass the RHCSA exam but will also make you a significantly more effective Linux system administrator in your daily work. The ability to quickly extract, transform, and analyze text data is a skill that you will use every single day in a production Linux environment.

Chapter 2: File and Archive Manipulation Drills

Working with files and archives is one of the most fundamental and frequently tested skill areas on the RHCSA EX200 exam. Every system administrator, regardless of experience level, spends a significant portion of their daily work creating, copying, moving, compressing, and extracting files and directories. The RHCSA exam expects you to perform these tasks quickly, accurately, and without hesitation. This chapter is designed to take you beyond the basics and into the kind of practical, hands-on drills that will build the muscle memory you need to succeed on exam day and in real-world production environments.

Throughout this chapter, you will work through progressively challenging exercises that cover file manipulation commands, hard and soft links, archive creation and extraction with `tar`, file compression using multiple utilities, and the critical skill of finding files based on various criteria. Each section includes detailed explanations, professional examples, and notes that clarify the nuances that often trip up exam candidates.

Before diving into the exercises, let us establish a solid understanding of the core commands and concepts you will be working with throughout this chapter.

Understanding the Linux File System and Basic File Operations

In Red Hat Enterprise Linux, everything is treated as a file. Regular files, directories, device nodes, sockets, and pipes are all represented within the file system hierarchy. The RHCSA exam tests your ability to navigate this hierarchy and manipulate its contents efficiently using command-line tools.

The following table provides a comprehensive reference of the primary file manipulation commands you must master for the exam.

Command	**Purpose**	**Common Options**	**Example**
`cp`	Copy files and directories	`-r` (recursive), `-p` (preserve attributes), `-a` (archive mode), `-v` (verbose)	`cp -a /source /destination`
`mv`	Move or rename files and directories	`-f` (force), `-i` (interactive), `-v` (verbose)	`mv oldname.txt newname.txt`
`rm`	Remove files and directories	`-r` (recursive), `-f` (force), `-i` (interactive)	`rm -rf /tmp/test-dir`
`mkdir`	Create directories	`-p` (create parent directories), `-m` (set permissions)	`mkdir -p /data/projects/2024`
`touch`	Create empty files or update timestamps	`-t` (set specific timestamp), `-a` (access time only)	`touch newfile.txt`
`ln`	Create links	`-s` (symbolic link), no option creates hard link	`ln -s /etc/hosts /tmp/hosts_link`
`find`	Search for files based on criteria	`-name`, `-type`, `-size`, `-user`, `-perm`, `-exec`	`find / -name "*.conf" -type f`
`tar`	Archive files	`-c` (create), `-x` (extract), `-t` (list), `-z` (gzip), `-j` (bzip2), `-J` (xz), `-f` (file)	`tar czf archive.tar.gz /data`
`gzip`	Compress files using gzip algorithm	`-d` (decompress), `-k` (keep original), `-v` (verbose)	`gzip largefile.log`
`bzip2`	Compress files using bzip2 algorithm	`-d` (decompress), `-k` (keep original), `-v` (verbose)	`bzip2 largefile.log`

`xz`	Compress files using xz algorithm	`-d` (decompress), `-k` (keep original), `-v` (verbose)	`xz largefile.log`
`stat`	Display detailed file information	No commonly needed options for RHCSA	`stat /etc/passwd`
`file`	Determine file type	No commonly needed options for RHCSA	`file /bin/bash`

Note: The `-a` option with `cp` is equivalent to `-dR --preserve=all` and is the most reliable way to copy directories while preserving all attributes including ownership, permissions, timestamps, and symbolic links. This is extremely important when performing backup operations or migrating data between systems, and it is a detail the RHCSA exam may test you on indirectly.

Working with Hard Links and Symbolic Links

Links are a concept that many RHCSA candidates find confusing at first, but they are straightforward once you understand the underlying mechanics. In the Linux file system, every file has an inode, which is a data structure that stores metadata about the file such as permissions, ownership, timestamps, and pointers to the actual data blocks on disk. The filename you see in a directory listing is simply a pointer to an inode.

A hard link is an additional directory entry that points to the same inode as the original file. Because both the original filename and the hard link point to the same inode, they are essentially indistinguishable from each other. Deleting one does not affect the other. The data is only removed from disk when the last hard link to an inode is deleted. Hard links cannot span file systems and cannot be created for directories.

A symbolic link, also called a soft link, is a special file that contains the path to another file or directory. It is similar to a shortcut in other operating systems. If the target file is deleted, the symbolic link becomes a dangling link that points to nothing. Symbolic links can span file systems and can point to directories.

The following table summarizes the differences between hard links and symbolic links.

Characteristic	**Hard Link**	**Symbolic Link**
Inode	Same inode as target	Different inode from target
Cross file system	Not allowed	Allowed
Link to directories	Not allowed for regular users	Allowed
Effect of deleting target	Link still works, data preserved	Link becomes broken (dangling)
Command to create	`ln target linkname`	`ln -s target linkname`
Identified by `ls -l`	Same as regular file, link count increases	Shows `l` file type and arrow to target

Let us now work through a series of practical exercises that will build your proficiency with these concepts.

Exercise 1: Building a Directory Structure and Populating It with Files

This exercise simulates a common RHCSA task where you must create a specific directory structure and populate it with files that have particular characteristics.

Begin by creating a working environment for this chapter's exercises.

```
mkdir -p /lab/chapter2/{projects,archives,backups,logs}
```

The `-p` option is essential here because it creates all parent directories in the path if they do not already exist. Without `-p`, the command would fail if `/lab` did not already exist.

Now create a set of files with specific content inside the projects directory.

```
for i in $(seq 1 10); do echo "This is project file number $i
with sample data for testing purposes" > /lab/chapter2/projects/
project_$i.txt; done
```

Create some additional files of varying sizes to use in later exercises involving the `find` command and compression.

```
dd if=/dev/urandom of=/lab/chapter2/projects/largefile.bin bs=1M
count=50
dd if=/dev/urandom of=/lab/chapter2/projects/mediumfile.bin bs=1K
count=500
dd if=/dev/urandom of=/lab/chapter2/logs/syslog_sample.log bs=1K
count=200
```

The `dd` command is used here to create files of specific sizes. The `if` parameter specifies the input file (in this case `/dev/urandom` for random data), `of` specifies the output file, `bs` sets the block size, and `count` determines how many blocks to write. Understanding `dd` is valuable for the RHCSA exam as it appears in various contexts including disk operations and file creation.

Verify the structure you have created.

```
ls -lR /lab/chapter2/
```

Note: The `-R` option with `ls` performs a recursive listing, showing all files and subdirectories within the specified path. This is a quick way to verify that a directory structure matches your expectations.

Exercise 2: Practicing with Hard Links and Symbolic Links

Create a hard link to one of the project files.

```
ln /lab/chapter2/projects/project_1.txt /lab/chapter2/projects/
project_1_hardlink.txt
```

Now verify that both files share the same inode number.