Chapter 4
Schedule Future Tasks

2. Schedule Future Tasks

Scheduling Deferred User Jobs

Understanding Deferred User Tasks

There are occasions when you may need to execute commands at a specific future time. For example, a user might schedule a maintenance operation to run overnight, or a system administrator could set up a temporary firewall configuration with an automated rollback in ten minutes, canceling the rollback if everything works as expected. These planned executions are referred to as deferred jobs, meaning they are scheduled for future execution.

One of the most efficient ways to handle deferred tasks on Red Hat Enterprise Linux is by using the at command, which is pre-installed and enabled by default. The at package includes the atd system daemon and commands like at and atq to interact with it.

Users can schedule jobs for execution by the atd daemon using the at command. The daemon organizes jobs into 26 priority-based queues, labeled from a to z, with jobs in later queues (z, y, etc.) having lower system priority.

Scheduling Deferred Jobs

To schedule a command for future execution, use the at TIMESPEC command. The at command reads commands from standard input (keyboard) and executes them at the designated time. When entering commands manually, complete input using Ctrl+D on an empty line. Alternatively, you can redirect input from a script file using:

```
at now +5 minutes < myscript
```

This schedules the script myscript to run in five minutes, avoiding manual command input.

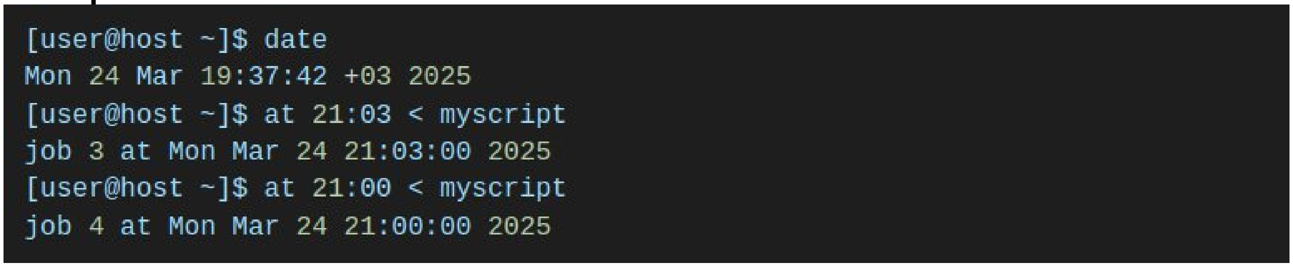
Using Time Specifications (TIMESPEC)

The at command accepts natural language-style time specifications for scheduling jobs. Some valid formats include:

- now +5min
- teatime tomorrow (equivalent to 16:00)
- noon +4 days
- 5pm August 3 2025

If only a time is provided, the command runs at the next occurrence of that time. If only a date is given, execution occurs at the current time on that date.

Example:



```
[user@host ~]$ date
Mon 24 Mar 19:37:42 +03 2025
[user@host ~]$ at 21:03 < myscript
job 3 at Mon Mar 24 21:03:00 2025
[user@host ~]$ at 21:00 < myscript
job 4 at Mon Mar 24 21:00:00 2025
```

To Schedule a job to run in two minutes from now. Save the output of the date command to the /home/user/myjob.txt file.
Pass the date >> /home/user/myjob.txt string as the input to the at command, so that the job runs in two minutes from now.

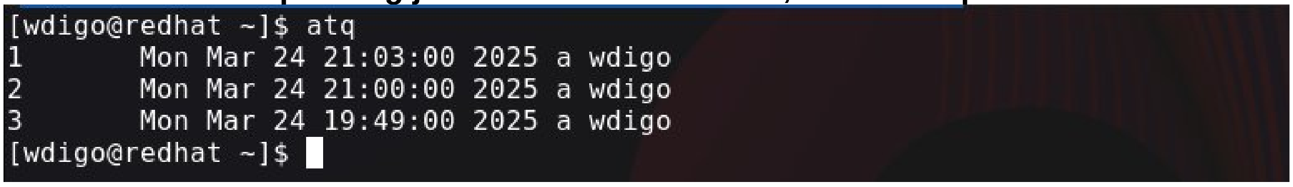
```
$ echo "date >> /home/user/myjob.txt" | at now +2min
```

To Interactively schedule a job in the g queue that runs at teatime (16:00). The job should print the “It's teatime” message to the /home/user/tea.txt file and Append the new messages to the /home/user/tea.txt file.

```
$ at -q g teatime
warning: commands will be executed using /bin/sh
at> echo "It's teatime" >> /home/user/tea.txt
at> Ctrl+d
$ atq
$ at –c JOBNUMBER
```

Viewing and Managing Deferred Jobs

To see a list of all pending jobs for the current user, use the atq command:



```
[wdigo@redhat ~]$ atq
1       Mon Mar 24 21:03:00 2025 a wdigo
2       Mon Mar 24 21:00:00 2025 a wdigo
3       Mon Mar 24 19:49:00 2025 a wdigo
[wdigo@redhat ~]$ 
```

Each line represents a scheduled job with:

- A unique job ID (1, 2, etc.).
- The execution date and time.
- The queue (a, etc.), where a is the default.
- The username of the job owner.

Note: Unprivileged users can only manage their own jobs, while the root user has control over all jobs on the system.

To inspect the details of a scheduled job, use:

```
at -c JOBNUMBER
```

This displays the job's execution environment and commands.

Canceling a Scheduled Job

To remove a scheduled job, use the atrm command:

```
atrm JOBNUMBER
```

For instance, if a remote firewall configuration succeeds and no rollback is needed, the associated scheduled job should be removed to prevent unnecessary changes.

References

- **man at(1), man atd(8)**
- **/usr/share/doc/at/timespec**

By understanding and using the at command effectively, you can efficiently schedule and manage future tasks, automating various system operations with precision.

Scheduling Recurring User Jobs

Understanding Recurring User Jobs

Recurring jobs are tasks that run repeatedly at predefined times. On Red Hat Enterprise Linux, the crond daemon, which is active by default, manages these scheduled jobs. The daemon reads scheduling instructions from user-specific crontab files and system-wide configuration files. Users can modify their personal crontab using the crontab -e command. If a scheduled job generates output and redirection isn't specified, crond will send the output via email to the job owner.

Setting Up Recurring Jobs

The crontab command is used to create, modify, and manage scheduled jobs. Below are some key crontab commands:

Command	Function
`crontab -l`	Displays the current user's scheduled jobs.
`crontab -r`	Removes all scheduled jobs for the user.
`crontab -e`	Opens the crontab file for editing.
`crontab filename`	Replaces all existing jobs with those from a specified file. If no file is provided, input is read from `stdin`.

Administrators can use the -u option to manage jobs for other users. However, using crontab as the root user is discouraged for security reasons. Privileged jobs should be managed differently, as explained in system job scheduling sections.

Crontab File Structure and Formatting

When editing the crontab file with crontab -e, the default editor is vim, unless another editor is set via the EDITOR environment variable. Each job occupies a separate line in the file. Recommended formatting includes:

- **Empty lines to improve readability.**
- **Comments, indicated by #, to provide descriptions.**
- **Environment variables, written as NAME=value, to configure settings affecting all subsequent jobs.**
    - **Common variables include:**
        - **SHELL: Specifies the shell interpreter.**
        - **MAILTO: Determines the email recipient for job output.**

Note: Sending email notifications requires additional system configuration, such as setting up an SMTP relay or local mail service.

Crontab Time and Date Format

A crontab entry consists of five scheduling fields followed by the command to execute:

```
<Minute> <Hour> <Day-of-Month> <Month> <Day-of-Week> <Command>
```

Scheduling rules:

- *** → Executes the job for all possible values of the field.**
- **x-y → Runs the job within a specified range of values.**
- **x,y → Executes the job for multiple specified values.**
- ***/x → Runs the job at intervals of x.**
- **Days of the week: Represented numerically (0 = Sunday, 1 = Monday, etc.), with 7 also considered Sunday.**
- **Months and days of the week can also be specified using three-letter abbreviations (Jan, Feb, Mon, Tue, etc.).**
- **Commands containing % must escape it or use it as a newline separator for input redirection.**

Examples of Recurring Jobs

1. **Annual Backup Job**
   Runs /usr/local/bin/yearly_backup at 9:00 AM on February 3rd every year.

```
0 9 3 2 * /usr/local/bin/yearly_backup
```

2. **Hourly Notification on Fridays in July**
   Sends an email with "Chime" every five minutes between 9:00 AM and 4:55 PM on Fridays in July.

```
*/5 9-16 * Jul 5 echo "Chime"
```

   Explanation: The job runs every 5 minutes within the 9 AM - 4 PM range. The last execution occurs at 4:55 PM. Since cron sends output via email if not redirected, the job owner will receive an email for each execution.

3. **Weekday Report Generation**
   Executes /usr/local/bin/daily_report two minutes before midnight (11:58 PM) from Monday to Friday.

```
58 23 * * 1-5 /usr/local/bin/daily_report
```

4. **Automated Email Check-In**
   Sends an email with the subject "Checking in" to developer@example.com at 9:00 AM on weekdays.

```
0 9 * * 1-5 mutt -s "Checking in" developer@example.com % Hi there, just checking in.
```

For additional time formats and configuration details, consult the following:

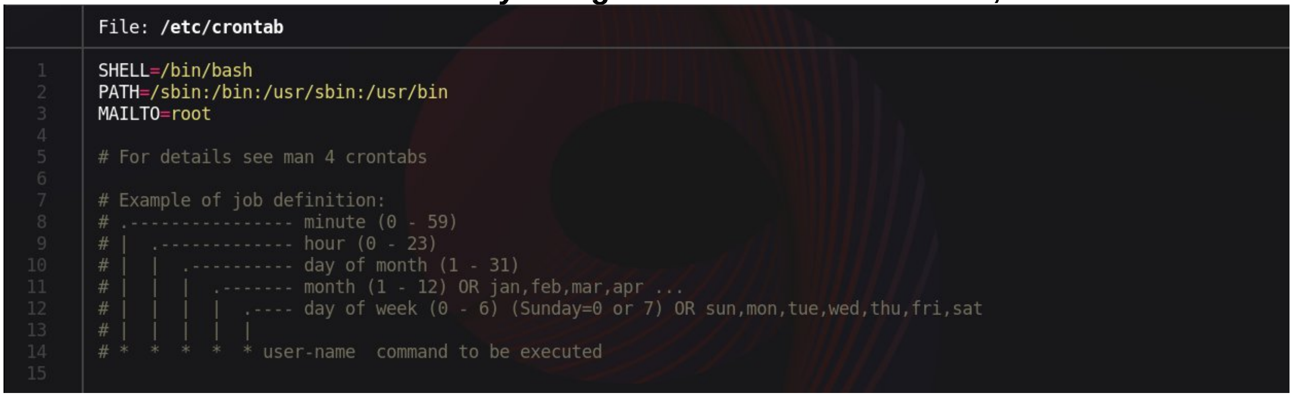
- **man 8 crond**
- **man 1 crontab**
- **man 5 crontab**

Automating Recurring System Tasks

Managing Recurring System Tasks

System administrators frequently need to automate tasks that run on a regular schedule. It is generally recommended to execute these tasks from system accounts instead of individual user accounts. Rather than using the crontab command for scheduling, administrators should utilize system-wide crontab files, which include an additional field specifying the user who will execute the job.

The /etc/crontab file includes a syntax guide within its comments, as shown below:



File: **/etc/crontab**

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root

# For details see man 4 crontabs

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name  command to be executed

```

To maintain system integrity, always place custom crontab files in the /etc/cron.d/ directory instead of modifying /etc/crontab, preventing potential overwrites from package updates. Software packages that require scheduled tasks also store their cron files in this directory, allowing administrators to group related jobs.

Additional Cron Directories

Besides individual crontab files, the cron system provides designated directories for scripts that need to run at predefined intervals:

- **/etc/cron.hourly/ – Runs hourly jobs**
- **/etc/cron.daily/ – Runs daily jobs**
- **/etc/cron.weekly/ – Runs weekly jobs**
- **/etc/cron.monthly/ – Runs monthly jobs**

These directories contain shell scripts rather than crontab files. Ensure scripts are executable by running:

```
chmod +x script_name
```

Running Periodic Commands with Anacron

Anacron is designed to guarantee that scheduled jobs execute even if the system was powered off during the intended runtime. It reads from the /etc/anacrontab file, which defines delayed execution parameters. When a job cannot run at the scheduled time, it will execute upon system startup.

Syntax of /etc/anacrontab

Each line consists of four fields:

1. **Period in days – Specifies how often the job should run (e.g., @daily for every day, @weekly for every week).**
2. **Delay in minutes – Defines the wait time before execution.**
3. **Job identifier – A label for logging purposes.**
4. **Command – The script or command to execute.**

Anacron maintains timestamps in /var/spool/anacron/, allowing administrators to verify when jobs were last executed. The START_HOURS_RANGE variable in /etc/anacrontab ensures jobs only run within a specified time window.

Automating Tasks with Systemd Timers

Systemd provides timer units that activate corresponding service units based on predefined schedules. These timers log execution events for easier troubleshooting.

Example: Systemd Timer Unit

The sysstat-collect.timer unit runs every 10 minutes to gather system statistics. Its configuration file (/usr/lib/systemd/system/sysstat-collect.timer) contains:

```
[Unit]
Description=Run system activity accounting tool every 10 minutes

[Timer]
OnCalendar=*:00/10

[Install]
WantedBy=sysstat.service
```

Defining Custom Intervals

Systemd timers support complex schedules. For example:

```
OnCalendar=2025-06-* 12:35,37,39:16
```

This runs the associated service at 12:35:16, 12:37:16, and 12:39:16 every day in June 2025.

Alternatively, relative scheduling can be used:

```
OnUnitActiveSec=15min
```

This triggers the service 15 minutes after the last execution.

Best Practices for Systemd Timer Management

Avoid modifying system timer files in /usr/lib/systemd/system/. Instead, copy them to /etc/systemd/system/ and make changes there. If duplicate names exist in both locations, systemd prioritizes the version in /etc/systemd/system/.

After modifying a timer unit, reload the daemon to apply changes:

```
systemctl daemon-reload
```

Enable and start the timer using:

```
systemctl enable --now <unitname>.timer
```

References

For further details, consult the following man pages:

- **crontab(5)**
- **anacron(8)**
- **anacrontab(5)**
- **systemd.time(7)**
- **systemd.timer(5)**
- **crond(8)**

Managing Temporary Files in Linux

Efficient management of temporary files is crucial for maintaining system performance and preventing disk space issues. This section covers how to enable and disable systemd timers and configure a timer to handle temporary files.

Understanding Temporary Files

- **Applications and services use temporary directories like /tmp for transient data.**
- **Some applications store temporary data in volatile directories under /run, which exist only in memory and are self-cleaning upon system reboot.**
- **Properly managing temporary files prevents issues like excessive disk usage or missing files required by daemons and scripts.**

Using systemd-tmpfiles for Temporary File Management

- **Red Hat Enterprise Linux provides the systemd-tmpfiles tool to manage temporary files and directories systematically.**
- **At system boot, the systemd-tmpfiles-setup service executes systemd-tmpfiles --create or --remove command, reading instructions from configuration files located in:**
  - **/usr/lib/tmpfiles.d/*.conf**
  - **/run/tmpfiles.d/*.conf**
  - **/etc/tmpfiles.d/*.conf**
- **These configuration files dictate which files and directories should be created, deleted, or secured.**

Automated Cleanup with systemd Timers

- **To prevent disks from filling up with old temporary data, the systemd-tmpfiles-clean.timer unit automatically triggers the systemd-tmpfiles-clean.service, executing systemd-tmpfiles --clean.**
- **The timer configuration includes:**
  - **OnBootSec=15min → Runs 15 minutes after system boot.**
  - **OnUnitActiveSec=1d → Repeats every 24 hours.**
- **Modifying the timer settings allows customization (e.g., running cleanup every 30 minutes).**
- **Reload systemd after changes with:**

```
systemctl daemon-reload
```

Manual Cleanup of Temporary Files

- **The command systemd-tmpfiles --clean removes files that haven't been accessed, changed, or modified within the configured time limit.**
- **Configuration files define cleanup rules, using syntax that includes:**
  - **Type (e.g., d for directory creation, D for directory cleanup).**
  - **Path (location of the file/directory).**
  - **Permissions (mode, ownership, and age settings).**

- **Example configuration:**

```
d /run/systemd/seats 0755 root root -
D /home/student 0700 student student 1d
L /run/fstablink - root root - /etc/fstab
```

  - **The first line ensures /run/systemd/seats exists with specific permissions.**
  - **The second line removes files in /home/student that haven't been accessed in over a day.**
  - **The third line creates a symbolic link without purging it.**

Configuration File Precedence

- **The system prioritizes configuration files in this order:**
  1. **/etc/tmpfiles.d/*.conf (highest priority, used for custom configurations).**
  2. **/run/tmpfiles.d/*.conf (used by daemons for runtime files).**
  3. **/usr/lib/tmpfiles.d/*.conf (vendor-provided defaults; should not be modified).**
- **To override vendor settings, copy the necessary file to /etc/tmpfiles.d/ and modify it.**

Best Practices

- **When testing configurations, apply commands from one file at a time.**
- **To apply changes for a specific configuration file, use:**

```
systemd-tmpfiles --clean /etc/tmpfiles.d/myconfig.conf
```

For more details, refer to the systemd-tmpfiles(8), tmpfiles.d(5), and systemd.timer(5) manual pages.