Week 04

IT-230

- Systemd Services
- Deferred Jobs with `at`
- Recurring User Jobs
- Recurring System Jobs
- Anacron
- Systemd Timers

IT-230 · Linux Administration

Start `workstation` and `servera` on your VDI

IT-230 · Linux Administration

Systemd Services

The first process, and everything it starts

What Systemd Does

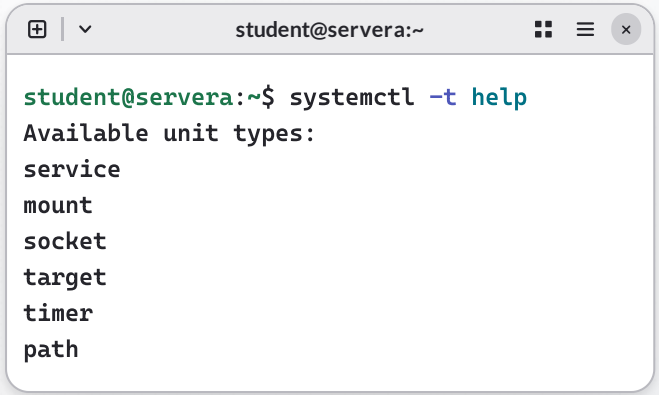
Systemd is **PID 1**, the first process the kernel starts

- Starts services
- Understands dependencies, so a service that needs the network waits for it
- Starts independent services **in parallel**, so boot is faster

> **Note**
>
> A **daemon** is a process running in the background. A **service** is what systemd calls the unit that manages it.

Everything Is a Unit

A unit is anything systemd knows how to manage



- `.service` starts a process
- `.mount` mounts a file system
- `.socket` listens, then starts a service on demand
- `.target` groups other units
- `.timer` starts a unit on a schedule

Where Units Live

root@servera:~

```
root@servera:~# ls /usr/lib/systemd/system/crond.service
root@servera:~# ls /etc/systemd/system/
```

`/usr/lib/systemd/system/` belongs to the package, and an update **overwrites it**

`/etc/systemd/system/` is yours, and it **takes precedence**

> **Warning**
>
> Systemd does not notice edits on its own. Run `systemctl daemon-reload` after changing any unit file.

Inside a Service Unit

student@servera:~

```
student@servera:~$ systemctl cat crond.service
# /usr/lib/systemd/system/crond.service
[Unit]
Description=Command Scheduler
After=auditd.service nss-user-lookup.target systemd-user-sessions.service

[Service]
ExecStart=/usr/sbin/crond -n $CRONDARGS
ExecReload=/bin/kill -URG $MAINPID
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

`[Unit]` says what and when, `[Service]` says how to run it, `[Install]` says where it belongs at boot

A Target Is a Group of Units

A target is a named set of units the system brings up together

`multi-user.target` is the normal running state of a server

`WantedBy=multi-user.target` means **start me when the system reaches that state**

> **Note**
>
> That is all you need for now. A later week digs into targets, boot order, and switching between them.

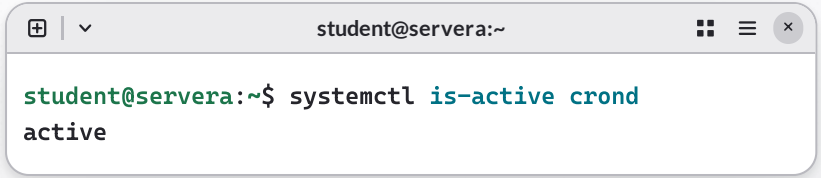
Reading `systemctl status`

```
student@servera:~$ systemctl status crond.service
● crond.service - Command Scheduler
     Loaded: loaded (/usr/lib/systemd/system/crond.service; enabled; preset: enabled)
     Active: active (running) since Tue 2026-03-24 12:02:34 UTC; 5 months 13 days ago
   Main PID: 853 (crond)
     CGroup: /system.slice/crond.service
             └─853 /usr/sbin/crond -n
Sep 06 13:01:01 servera.lab.example.com anacron[12866]: Anacron started on 2026-09-06
```

The newest journal entries, without running a second command

Running and Enabled?

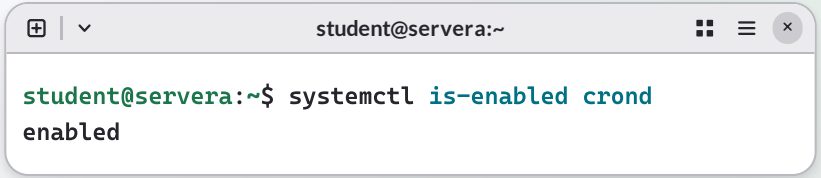
Is it running now?



`active` - Currently running

`inactive` - **Not** currently running

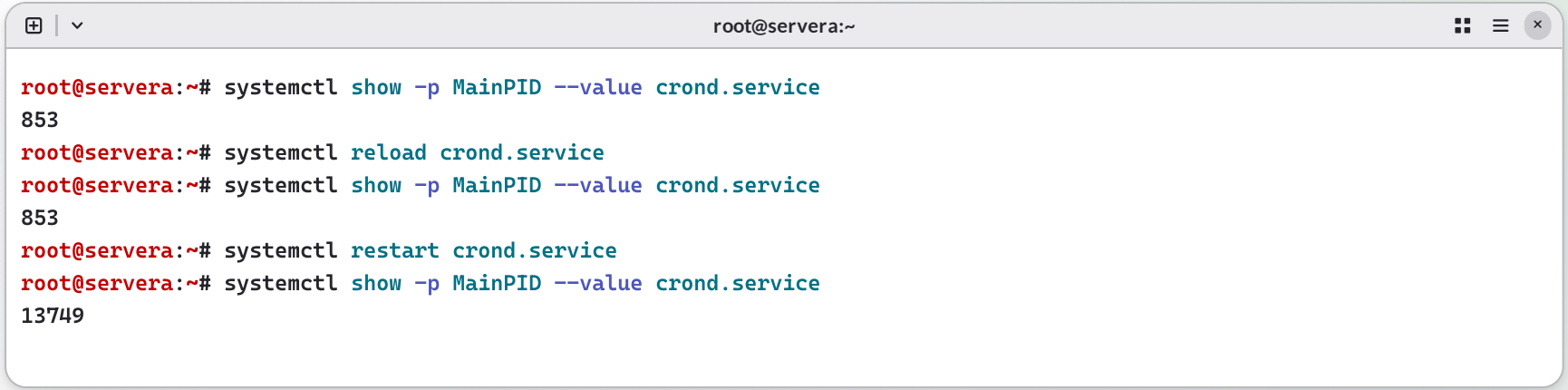
Will it start at boot?



`enabled` - Started at boot

`disabled` - **Not** started at boot

Start, Stop, Restart, Reload



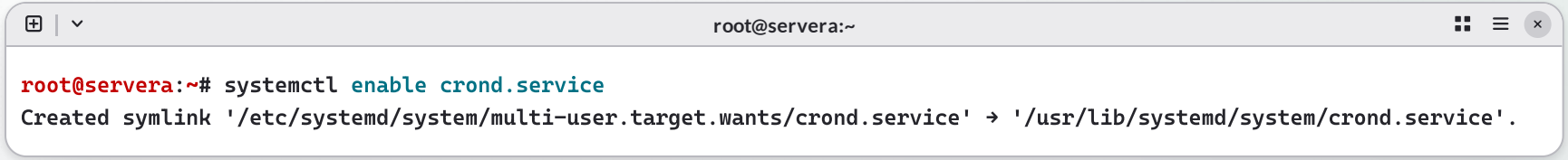
`reload` re-reads the config and keeps the same process

`restart` stops and starts a **new** one

Start Now and at Boot

`systemctl enable {SERVICE} --now` is equivalent to

`systemctl start {SERVICE} && systemctl enable {SERVICE}`



Enabling creates a symlink into the target's `wants` directory specified in `WantedBy=`

Masked Cannot Start

Masking points a unit at `/dev/null` so nobody can start it, on purpose or by accident

root@servera:~

```
root@servera:~# systemctl mask httpd.service
Created symlink '/etc/systemd/system/httpd.service' → '/dev/null'.
root@servera:~# systemctl start httpd.service
Failed to start httpd.service: Unit httpd.service is masked.
root@servera:~# systemctl unmask httpd.service
Removed '/etc/systemd/system/httpd.service'.
```

A **disabled** unit can be started manually

A **masked** unit will not start at all.

Running Something First

`ExecStartPre` runs first, and if it fails then `ExecStart` never runs and the unit is marked **failed**

man-db-restart-cache-update.service

```
1 ExecStartPre=/usr/bin/rm -rf /var/cache/man/*
2 ExecStart=/usr/bin/systemd-run /usr/bin/systemctl start man-db-cache-update.service
```

student@servera:~

```
student@servera:~$ man systemd.directives
       ExecStart=
           systemd.service(5)
       ExecStartPre=
           systemd.service(5), systemd.socket(5)
       ExecStartPost=
           systemd.service(5), systemd.socket(5)
       ExecStop=
           systemd.service(5)
       ExecStopPost=
           systemd.service(5), systemd.socket(5)
```

Hundreds more, each naming the page that documents it

HANDS-ON EXERCISE

Exercise: A Robust Sync Service

GOAL

Synchronize a directory when the remote host is reachable, or fail visibly when it is not

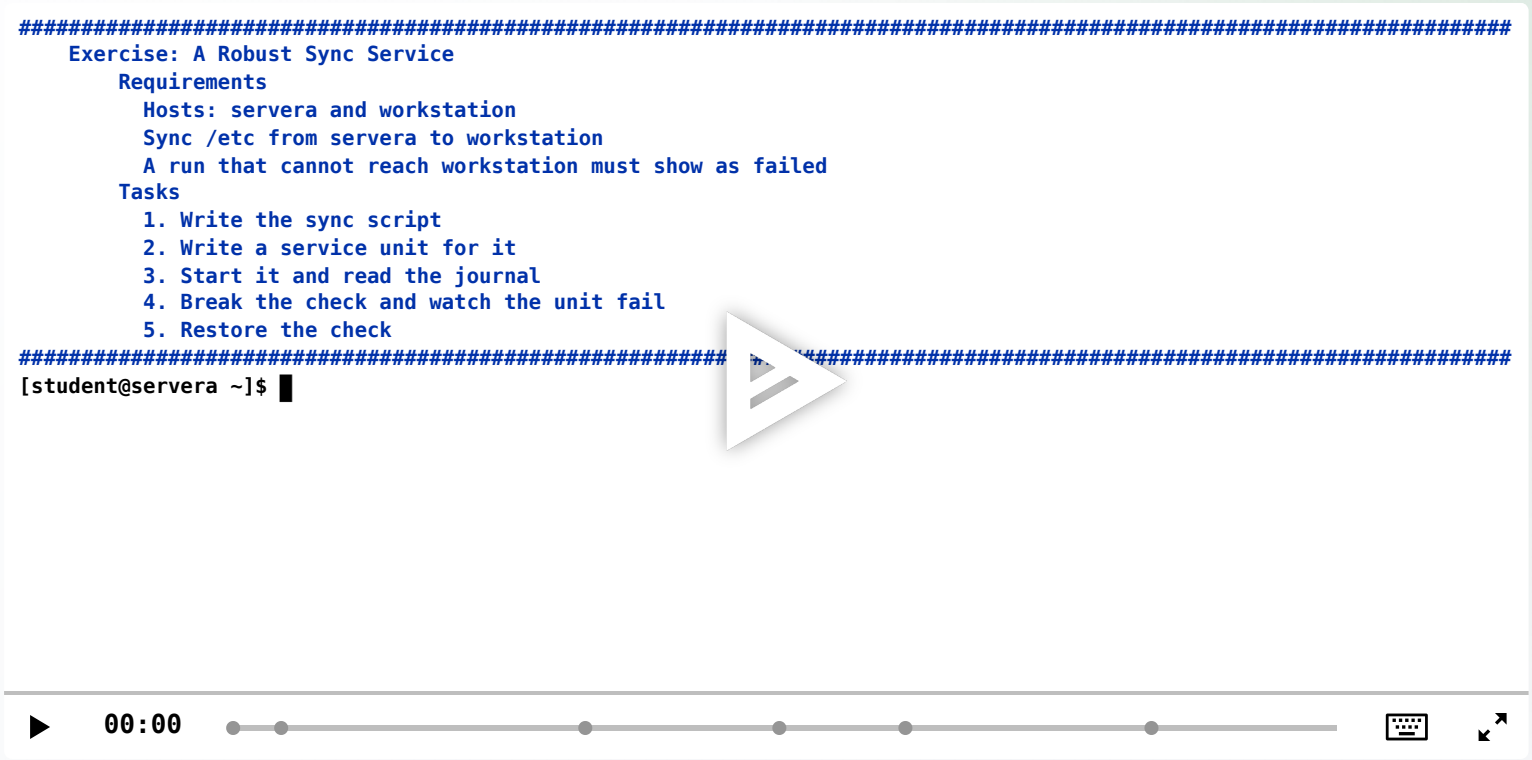
ENVIRONMENT

Hosts: `servera` and `workstation`

WORKFLOW

1. Write a script that syncs `/etc` to `workstation` and run it by hand
2. Write a `.service` unit that runs the script
3. Reload systemd, start the unit, and read its status and journal
4. Point the check at an unreachable address and watch the unit fail
5. Find it with `systemctl --failed`, then restore the check

Exercise: A Robust Sync Service



Written exercise

IT-230 · Linux Administration

Deferred Jobs with `at`

Run a one-off command at some future time

Do This, Later

Hand a command to `at` to run it later

- Good for **one-off** jobs that run once and are done
- Not for **recurring** jobs like every day at 8 p.m.

```
1 echo "reboot" | at now +2min
2 at 23:10 < database_backup.sh
```

> **Note**
>
> Reset a firewall in ten minutes, in case the rule you are about to write locks you out.
>
> ```
> 1 echo "firewall-cmd --reload" | at now +10min
> ```

Install `at`

The package is not part of a minimal RHEL installation

student@servera:~

```
student@servera:~$ sudo dnf install at
...output omitted...
Installing:
 at          x86_64          3.2.5-12.el10           dvd-baseos           70 k

Transaction Summary
================================================================================
Install  1 Package
...output omitted...
Is this ok [y/N]: y
...output omitted...
Created symlink '/etc/systemd/system/multi-user.target.wants/atd.service' → '/usr/lib/systemd/system/atd.service'.
...output omitted...
Complete!
```

Installing the package **enables** `atd`, but does not start it

Start `atd`

`at` only queues jobs, and `atd` is the daemon that runs them

student@servera:~

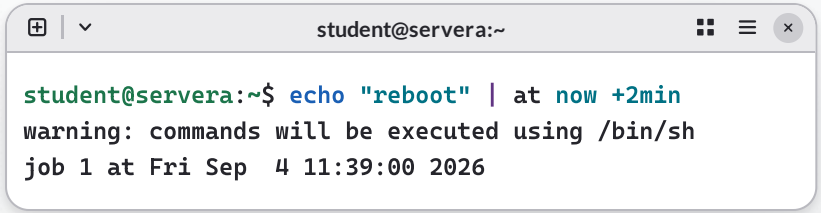
```
1  student@servera:~$ systemctl is-active atd
2  inactive
3  student@servera:~$ sudo systemctl start atd
4  student@servera:~$ systemctl is-active atd
5  active
```

> **Warning**
>
> A job scheduled while `atd` is stopped waits in the queue. It does not run on time.

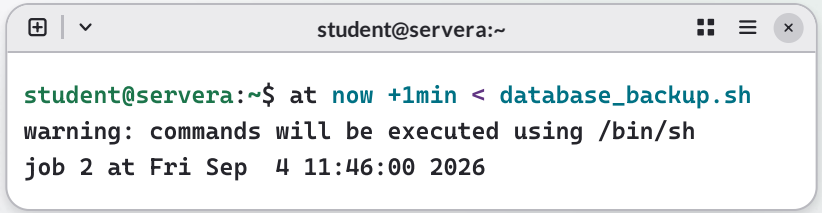
Two Ways to Hand Over a Command

`at` reads the commands to run from standard input

Pipe a command



Redirect a script



View the Queue

```
student@servera:~$ atq
1    Fri Sep  4 11:46:00 2026    a    student
```

The user it runs as

Inspect a Job

`at -c` prints the job as `atd` will run it, including the environment `at` saved for you

student@servera:~

```
student@servera:~$ at -c 1
#!/bin/sh
# atrun uid=1000 gid=1000
# mail student 0
umask 22
SHELL=/bin/bash; export SHELL
PWD=/home/student; export PWD
...many more environment variables omitted...
cd /home/student || {
     echo 'Execution directory inaccessible' >&2
     exit 1
}
${SHELL:-/bin/sh} << 'marcinDELIMITER31c8c648'
date --iso-8601=seconds >> /tmp/backup.log
marcinDELIMITER31c8c648
```

Remove a Job

`atrm` takes the job number from `atq`

student@servera:~

```
student@servera:~$ atq
1   Fri Sep  4 11:46:00 2026 a student
2   Fri Sep  4 12:36:00 2026 a student
student@servera:~$ atrm 2
student@servera:~$ atq
1   Fri Sep  4 11:46:00 2026 a student
```

Did It Run?

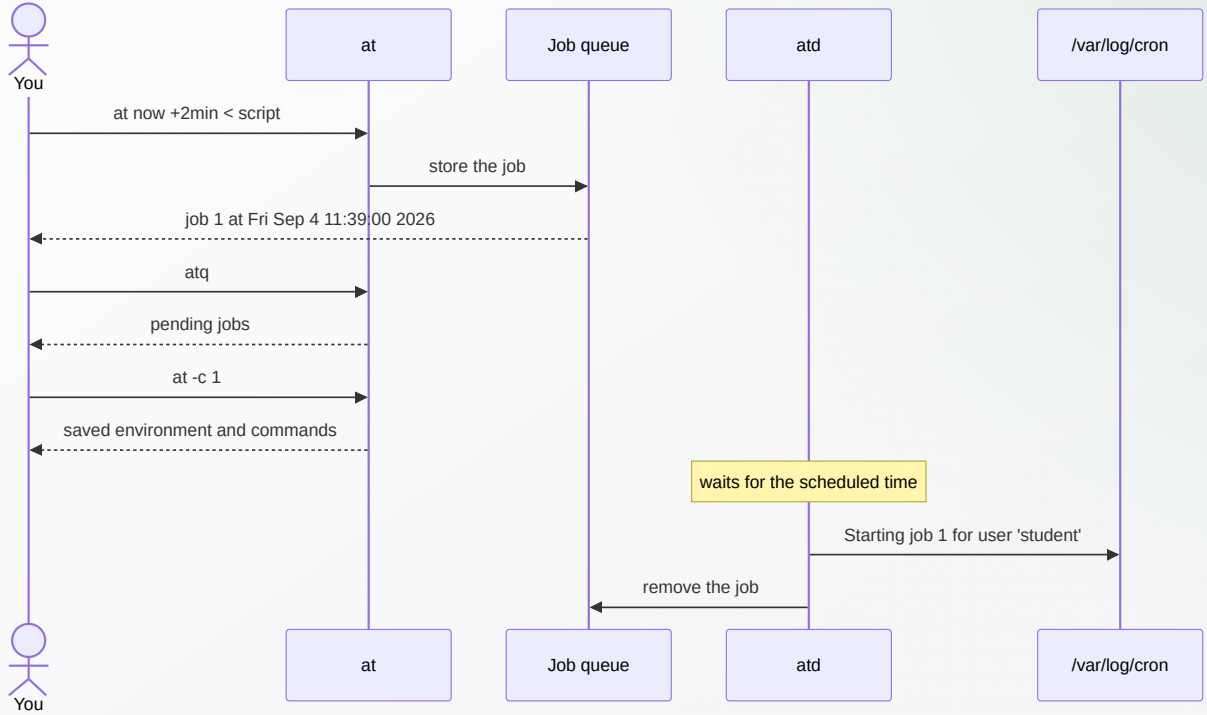
A job disappears from `atq` whether it succeeded or failed, so the log is the proof

student@servera:~

```
student@servera:~$ sudo less /var/log/cron
Sep  4 11:37:00 servera atd[10208]: Starting job 1 (a0000101c6d839) for user 'student' (1000)
```

> **Warning**
>
> `atd` logs that it started the job. Check the job's own output to know what it did.

The Whole Trip



HANDS-ON EXERCISE

Exercise: Finding Recently Modified Logs

GOAL

Schedule a search for recently changed log files and confirm that it ran

ENVIRONMENT

Host: `servera`

WORKFLOW

1. Install `at` and start `atd`
2. Build a `find` command that lists logs modified in the last two minutes
3. Schedule that command to run two minutes from now
4. Inspect the pending job with `atq` and `at -c`
5. Confirm in `/var/log/cron` that `atd` started the job
6. Verify the file the job wrote

Exercise: Finding Recently Modified Logs

```
################################################################################################################
    Exercise: Finding Recently Modified Logs
        Requirements
          Host: servera
          Record which files under /var/log change while you are away from the console
          The search runs once, two minutes from now
          Save the results to /tmp/log_audit
        Tasks
          1. Install at and start atd
          2. Build the search
          3. Schedule it two minutes from now
          4. Inspect the pending job
          5. Confirm the job ran
          6. Clean up
################################################################################################################
[student@servera ~]$
```

Written exercise

IT-230 · Linux Administration

Recurring User Jobs

Do this, later, and on repeat, as a specific user

Crontab

- Available on **every** Linux system
- Every user has one
- Finicky, ubiquitous syntax

> **Note**
>
> `at` runs a job once. `cron` runs it repeatedly on a schedule you define.

Five Fields, Then a Command

```
*  *  *  *  *  command to be executed
```

What runs when all five fields match

A `*` means every value of that field

Reading a Schedule

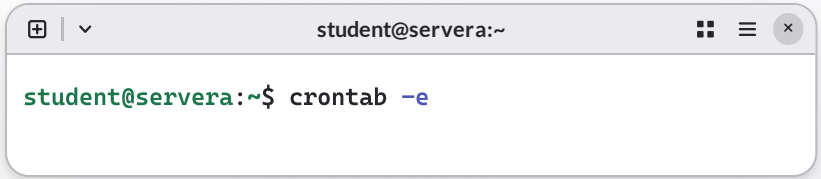
/etc/crontab

```
.----------------- 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
|  |  |  |  |
*  *  *  *  * command to be executed
```

```
# 2:15 a.m. on Sunday, Monday, and Tuesday
15 2 * * sun,mon,tue echo "This runs every Sunday, Monday and Tuesday at 2:15a.m." >> stating_the_obvious.log
```

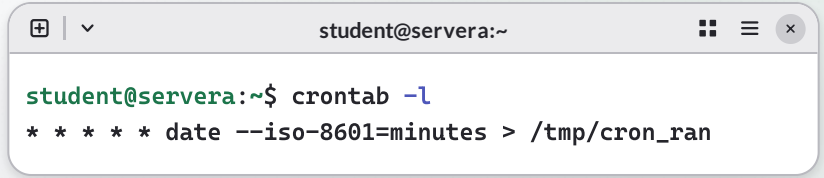
Editing and Reading Your Crontab

Edit



Opens in `$EDITOR` (`vi` unless you changed it)

Read



The file itself lives in `/var/spool/cron/`

Did It Run?

`crond` records every job it starts and finishes

student@servera:~

```
student@servera:~$ sudo grep "(student)" /var/log/cron
Sep  5 14:16:01 servera CROND[11393]: (student) CMD (date --iso-8601=seconds > /tmp/cron_ran)
Sep  5 14:16:01 servera CROND[11391]: (student) CMDEND (date --iso-8601=seconds > /tmp/cron_ran)
student@servera:~$ cat /tmp/cron_ran
2026-09-05T14:17:01+00:00
```

CMD is the job starting

CMDEND is it finishing

Removing Your Crontab

student@servera:~

```
1  student@servera:~$ crontab -r
2  student@servera:~$ crontab -l
3  no crontab for student
```

No confirmation

`crontab -r` deletes **every** entry you have, immediately. There is no undo and no prompt.

Cron's PATH Is Not Your PATH

`crond` never reads your login files, so your `PATH` is not the one your job gets

> **Warning**
>
> A script in `~/bin` or `/usr/local/bin` fails with `command not found`, and only the log says so. Use an absolute path, or set `PATH=` in the crontab.

Your shell

```
/home/student/.local/bin
/home/student/bin
/usr/local/bin
/usr/bin
/usr/local/sbin
/usr/sbin
```

Any cron job

```
/usr/bin
/bin
/usr/sbin
/sbin
```

HANDS-ON EXERCISE

Exercise: Backing Up Documents Nightly

GOAL

Schedule a nightly backup of your documents and prove that it ran

ENVIRONMENT

Host: `servera`

WORKFLOW

1. Create a small `~/Documents` tree worth backing up
2. Build and test the backup command by hand
3. Schedule it one minute out with `crontab -e`
4. Confirm in `/var/log/cron` that it ran, and check the backup
5. Reset the schedule to 2 a.m. and review it with `crontab -l`

Exercise: Backing Up Documents Nightly

```
################################################################################################################
    Exercise: Backing Up Documents Nightly
        Requirements
          Host: servera
          Back up /home/student/Documents every night at 2 a.m.
          Each run writes to its own timestamped directory under /tmp
        Tasks
          1. Create something worth backing up
          2. Build and test the backup command
          3. Schedule it one minute from now
          4. Confirm it ran
          5. Set the real schedule
          6. Clean up
################################################################################################################
[student@servera ~]$
```

Written exercise

IT-230 · Linux Administration

Recurring System Jobs

Do this, later, and on repeat, as a specified user

System Crontabs

One field more than your own crontab: the **user-name** a job runs as

/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
```

Service Accounts

Applications run under their own accounts, not under yours

```
student@servera:~
1  student@servera:~$ grep chrony /etc/passwd
2  chrony:x:995:994:chrony system user:/var/lib/chrony:/sbin/nologin
3  student@servera:~$ sudo su - chrony
4  This account is currently not available.
```

A **nologin** shell refuses every interactive login, so `chrony` can never run `crontab -e` for itself

Grouping Jobs in `/etc/cron.d`

`/etc/cron.d` is a drop-in directory (like `~/.bashrc.d`)

`crond` reads every file in the directory as a crontab

Keep related jobs together in one named file

/etc/cron.d/web_server_maintenance

```
1 0/15 * * * * apache /usr/bin/regenerate_static_web_content.sh # every 15 minutes starting a 0
2 0,30 * * * * apache /usr/bin/incremental_backup.sh # every 30 minutes
3 0 18 * * fri apache /usr/bin/full_backup.sh # Every Friday at 6p.m.
```

> **Warning**
>
> Put your jobs in `/etc/cron.d/` instead of `/etc/crontab`
>
> An update to the `crontabs` package update overwrites `/etc/crontab`.

Did It Run?

/etc/cron.d/patch_checks

```
0 6 * * *   root /usr/bin/check_updates.sh
0 7 * * mon root /usr/bin/report_pending_reboots.sh
```

student@servera:~

```
student@servera:~$ sudo grep check_updates /var/log/cron
Sep  5 16:19:01 servera CROND[12287]: (root) CMD (/usr/bin/check_updates.sh)
Sep  5 16:19:01 servera CROND[12286]: (root) CMDEND (/usr/bin/check_updates.sh)
```

The log says **(root)** instead of `(student)`, because the drop-in file named the user

HANDS-ON EXERCISE

Exercise: Backing Up `/etc` Nightly

GOAL

Back up `/etc` every night as `root`, from a system crontab rather than your own

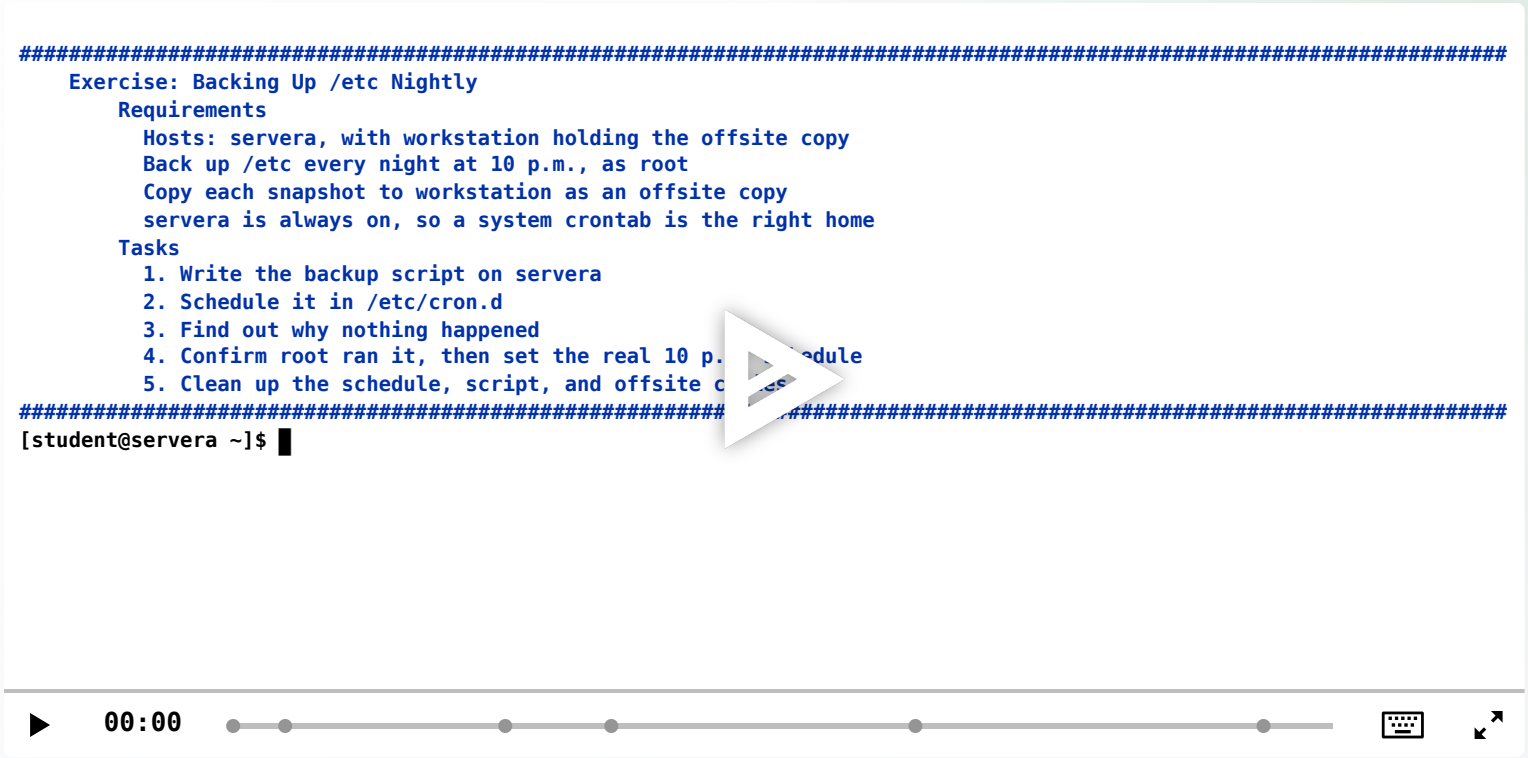
ENVIRONMENT

Hosts: `servera`, with `workstation` holding the offsite copy

WORKFLOW

1. Write a short backup script on `servera` and make it executable
2. Schedule it in `/etc/cron.d/` and test it every minute
3. Find out why nothing ran, then correct the entry
4. Confirm in `/var/log/cron` that `root` ran it, and check the copy on `workstation`
5. Set the real 10 p.m. schedule and read the entry back

Exercise: Backing Up `/etc` Nightly



Written exercise

IT-230 · Linux Administration

Anacron

The cron for desktops and laptops

Same Job, Different Machine

A crontab entry only fires if the machine is awake at that moment

Server

Awake at 6 a.m.

The update check runs on time

Laptop

Asleep at 6 a.m.

The check never happens, and cron never looks back

Anacron

Anacron asks has this run lately? instead of is it 6 a.m. yet?

Put `check_updates.sh` in one of these instead, as an executable script

- `/etc/cron.daily/`
- `/etc/cron.weekly/`
- `/etc/cron.monthly/`

If the machine was off when the job was due, anacron runs it once the machine is back

Where Anacron Keeps the Date

`/etc/anacrontab` lists each group: how many days between runs, how long to wait before starting, a name for the group, and the command

/etc/anacrontab

```
#period in days   delay in minutes   job-identifier   command
1   5   cron.daily      nice run-parts /etc/cron.daily
7   25  cron.weekly     nice run-parts /etc/cron.weekly
@monthly 45 cron.monthly        nice run-parts /etc/cron.monthly
```

`run-parts` runs every executable in the directory, which is why your script needs `chmod a+x`

The **job-identifier** is also a file under `/var/spool/anacron/` , holding the date that group last ran

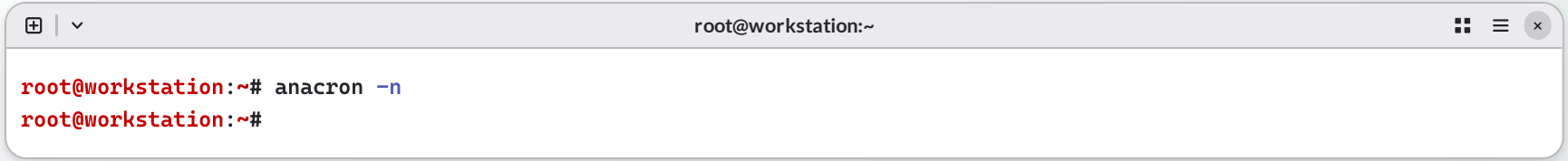
root@workstation:~

```
root@workstation:~# cat /var/spool/anacron/cron.daily
20260915
```

Running It On Demand

Cron starts anacron every hour from `/etc/cron.hourly/0anacron`, and anacron then waits out the delay before running anything

Test it with **`anacron -n`**, which runs whatever is due right now and ignores those delays



No output, and no job, because today's date is already in `/var/spool/anacron/cron.daily`

Anacron Tradeoffs

You give up three things

You do not choose the hour. The job runs as `root`. On a laptop it is skipped while running on battery.

Anacron tracks each directory by date, so it knows when a daily job is overdue

Nothing runs while the job is up to date, however often the machine wakes up

HANDS-ON EXERCISE

Exercise: Backing Up a Laptop with Anacron

GOAL

Give a daily backup to anacron on a machine that is asleep at night, then watch it catch up

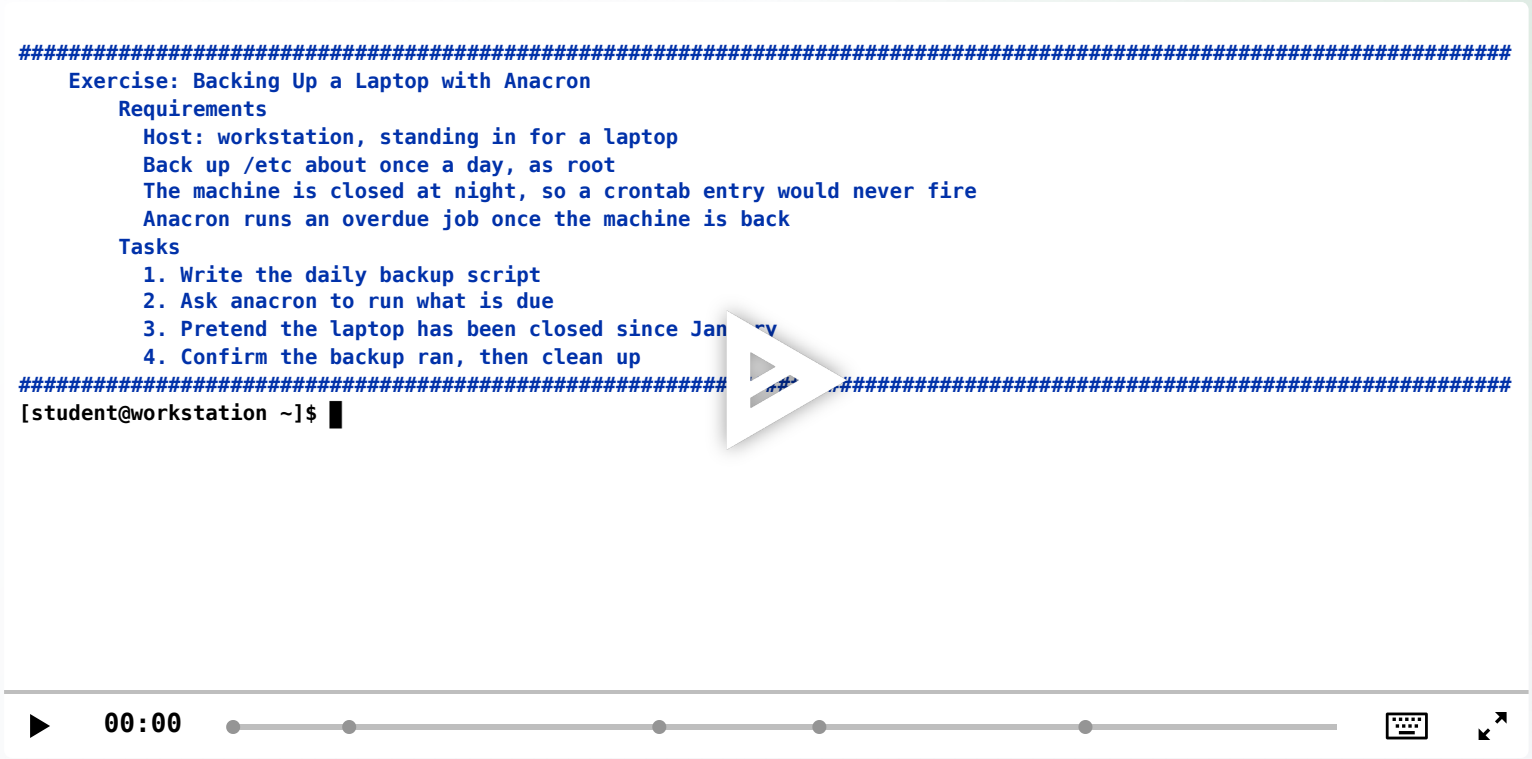
ENVIRONMENT

Host: `workstation`, standing in for a laptop

WORKFLOW

1. Write a daily backup script in `/etc/cron.daily/` and make it executable
2. Read the date anacron recorded for the daily jobs
3. Ask anacron to run what is due, and see that nothing is
4. Backdate that record so the daily jobs are months overdue
5. Ask again, then confirm the backup ran and the recorded date caught up

Exercise: Backing Up a Laptop with Anacron



Written exercise

IT-230 · Linux Administration

Systemd Timers

How RHEL schedules its own recurring work

It Takes Two Units

A timer says **when** and its service says **what**

Matching names link them, so you enable the timer, never the service

logrotate.timer

```
[Timer]
OnCalendar=daily
RandomizedDelaySec=1h
Persistent=true
```

logrotate.service

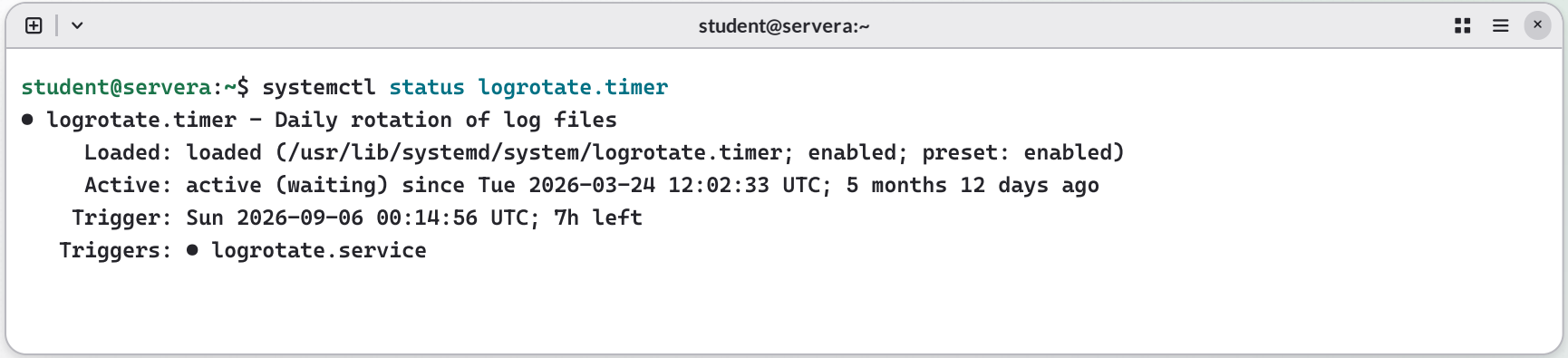
```
[Service]
Type=oneshot
ExecStart=/usr/sbin/logrotate /etc/logrotate.conf
```

Finding Timers

```
student@servera:~$ systemctl list-units -t timer
  UNIT                         LOAD   ACTIVE SUB     DESCRIPTION
  dnf-makecache.timer          loaded active waiting dnf makecache --timer
  logrotate.timer              loaded active waiting Daily rotation of log files
  systemd-tmpfiles-clean.timer loaded active waiting Daily Cleanup of Temporary Directories
student@servera:~$ systemctl list-unit-files -t timer
UNIT FILE                      STATE    PRESET
insights-client.timer          disabled disabled
logrotate.timer                enabled  enabled
```

`list-units` shows what is loaded now, `list-unit-files` shows everything installed

When Does It Fire Next?



Trigger is when it fires next, and **Triggers** is the service it will start

Saying When

The `[Timer]` section answers when, and it can count from several starting points

Option	Counts from
`OnCalendar`	a wall-clock date and time, such as `daily`
`OnBootSec`	when the machine booted
`OnStartupSec`	when systemd itself started
`OnActiveSec`	when the timer was activated
`OnUnitActiveSec`	the last time the service it triggers ran

Timers Catch Up Too

`Persistent=true` records the last run on disk

If the machine was off when the timer was due, it fires once the machine is back

> **Sound familiar?**
>
> This is what anacron does for `/etc/cron.daily`, except a timer does it per unit, without needing `root` and without the battery rule.

Writing Your Own

Never edit a unit in `/usr/lib/systemd/system`, because a package update overwrites it

root@servera:~

```
root@servera:~# vim /etc/systemd/system/etc-backup.service
root@servera:~# vim /etc/systemd/system/etc-backup.timer
root@servera:~# systemctl daemon-reload
root@servera:~# systemctl enable --now etc-backup.timer
```

`daemon-reload` makes systemd read your changes, and `--now` starts the timer as well as enabling it

Which One Do I Reach For?

All three are worth knowing, and cron and timers are both on the RHCSA exam

`at`

A job that runs **once**

cron

A recurring job you are writing yourself

systemd timer

What RHEL already ships its own jobs as

Also when you need a boot-relative schedule, or a unit dependency

HANDS-ON EXERCISE

Exercise: Scheduling a Timer Unit

GOAL

Schedule a job with a timer, then make it survive the machine being switched off

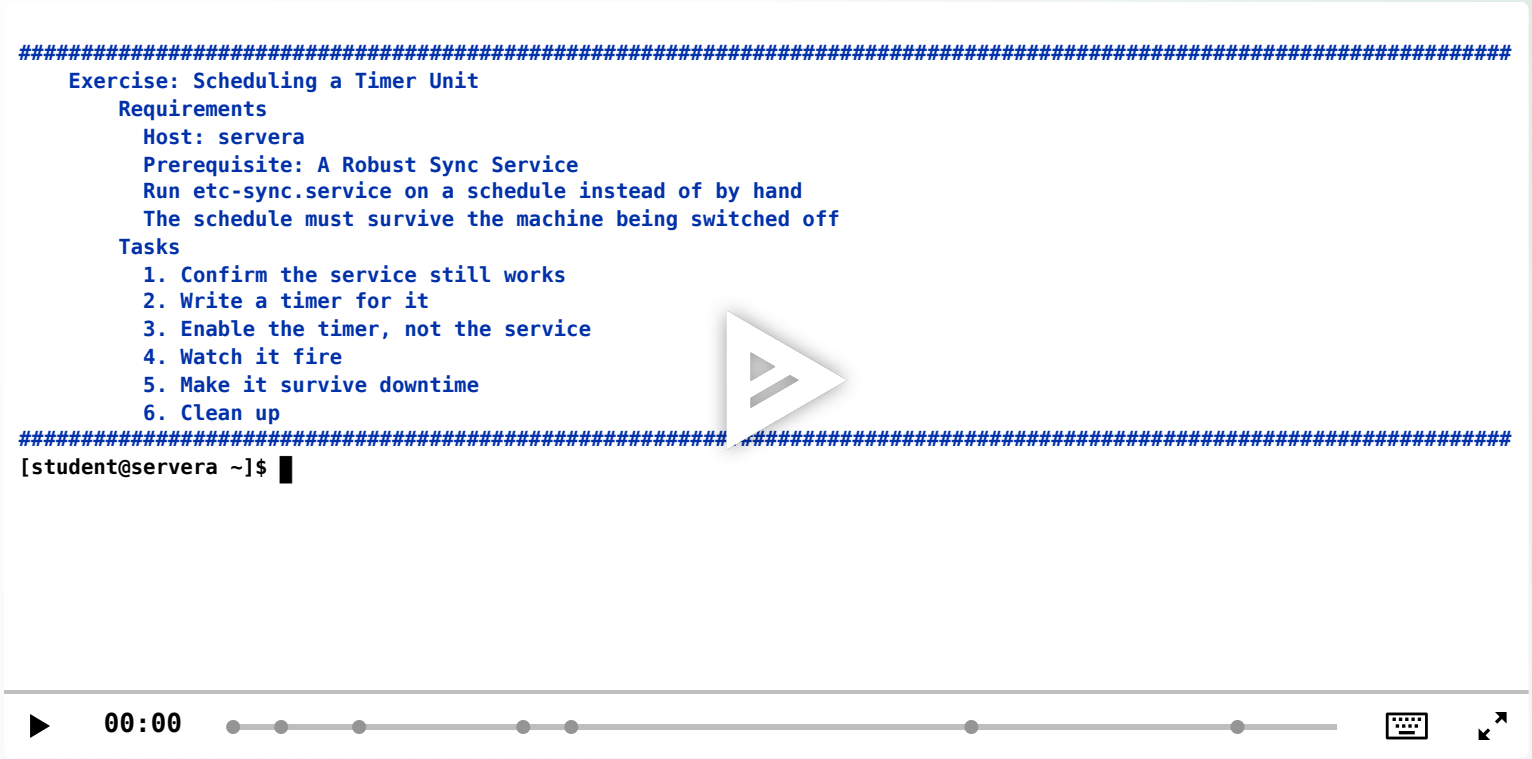
ENVIRONMENT

Host: `servera`

WORKFLOW

1. Write a script worth scheduling and run it once by hand
2. Write a `.service` unit that runs it
3. Write a matching `.timer` unit and enable it
4. Confirm it fires with `systemctl list-timers` and the journal
5. Add `Persistent=true` and reload, then check the timer again

Exercise: Scheduling a Timer Unit



Written exercise