Table of Contents

$ at

Intermediate

Schedule a one-time command to run at a specific time

at schedules a command to run once at a specified time in the future. Unlike cron which runs tasks repeatedly, at runs a one-time job. It is perfect for scheduling maintenance, delayed tasks, and one-off operations.

at reads commands from standard input or a file and queues them for later execut...

Options & Flags

`TIME`	Schedule job at specified time
`-l`	List pending jobs (same as atq)
`-d N`	Delete job N (same as atrm)
`-f`	Read commands from file
`-m`	Send mail when job completes

Practical Examples

Example: Schedule for tonight

```
$ echo "/usr/local/bin/backup.sh" | at 11:00 PM
job 5 at 2024-01-15 23:00
```

Schedules backup script for 11 PM tonight.

Example: In one hour

```
$ echo "systemctl restart nginx" | at now + 1 hour
```

Restarts nginx one hour from now.

Example: Tomorrow morning

```
$ at 8:00 AM tomorrow << EOF\n/usr/local/bin/report.sh\nEOF
```

Schedules a report for tomorrow morning using here-doc.

Example: From file

```
$ at -f maintenance.sh 2:00 AM Saturday
```

Reads and schedules commands from a file.

Example: List pending jobs

```
$ atq
5\t2024-01-15 23:00 a user
```

Shows all pending at jobs.

Tips & Best Practices

Pro Tip: Flexible time formats: "now + 30 minutes", "midnight", "noon tomorrow", "4pm next Friday", "teatime" (4pm), "2:00 AM 01/20/2025".

Note: Environment is preserved: at inherits the current environment (PATH, HOME, etc.) so commands work the same as if typed interactively.

Warning: atd must be running: at requires the atd daemon: sudo systemctl enable --now atd.

$ crontab

Intermediate

Schedule recurring tasks with cron

crontab schedules recurring tasks (cron jobs) to run automatically at specified times. Cron is the standard Linux task scheduler, used for backups, log rotation, reports, maintenance, and any periodic task.

Each user has their own crontab. The cron format specifies minute (0-59), hour (0-23), da...

Options & Flags

`-e`	Edit your crontab
`-l`	List your crontab
`-r`	Remove your crontab
`-u USER`	Edit another user crontab (root only)

Practical Examples

Example: Edit crontab

```
$ crontab -e
```

Opens crontab editor to add/modify scheduled tasks.

Example: List cron jobs

```
$ crontab -l
0 2 * * * /usr/local/bin/backup.sh\n*/5 * * * * /usr/local/bin/check_health.sh
```

Shows all scheduled tasks for current user.

Example: Every 5 minutes

```
$ */5 * * * * /path/to/script.sh
```

Runs every 5 minutes.

Example: Daily at 2 AM

```
$ 0 2 * * * /path/to/backup.sh
```

Runs at 2:00 AM every day.

Example: Weekly on Sunday

```
$ 0 3 * * 0 /path/to/weekly_report.sh
```

Runs at 3:00 AM every Sunday.

Tips & Best Practices

Pro Tip: Cron format: m h dom mon dow: minute hour day-of-month month day-of-week. * means every. */5 means every 5. 1,15 means 1st and 15th.

Warning: PATH is minimal: Cron uses minimal PATH. Always use full paths in cron jobs: /usr/local/bin/python3 instead of python3.

Note: Redirect output: Cron sends output as email. Redirect to log: >> /var/log/cron.log 2>&1. Or discard: > /dev/null 2>&1.

$ logger

Intermediate

Send messages to the system log

logger makes entries in the system log (syslog). It provides a way for shell scripts and commands to write messages to the standard logging infrastructure.

logger is the proper way for scripts to log messages. Instead of writing to custom log files, logger sends messages to rsyslog/journald wher...

Options & Flags

`-p`	Set priority (facility.level)
`-t`	Set tag (program name)
`-s`	Also output to stderr
`-i`	Include PID

Practical Examples

Example: Simple log message

```
$ logger "Backup completed successfully"
```

Writes a message to syslog.

Example: Tagged message

```
$ logger -t backup-script "Daily backup finished, 2.5GB compressed"
```

Logs with a tag for easy filtering.

Example: Error priority

```
$ logger -p user.err -t myapp "Failed to connect to database"
```

Logs an error-level message.

Example: In a script

```
$ #!/bin/bash\nlogger -t deploy "Deploy started"\n./deploy.sh && logger -t deploy "Deploy succeeded" || logger -t deploy -p user.e
```

Script that logs start, success, and failure.

Example: Verify message

```
$ logger -t test "Hello syslog" && journalctl -t test --no-pager -n 1
```

Logs a message and immediately checks it.

Tips & Best Practices

Pro Tip: Use tags for filtering: -t tag makes log messages filterable: journalctl -t backup-script shows only your messages.

Note: Priority levels: Priorities: emerg, alert, crit, err, warning, notice, info, debug. Default is user.notice.

Warning: Do not log secrets: Log messages are stored in plain text and may be forwarded to remote servers. Never log passwords or tokens.

$ logrotate

Intermediate

Rotate, compress, and manage log files

logrotate manages log file rotation, compression, and deletion. It prevents log files from growing indefinitely and consuming all disk space by automatically rotating, compressing, and removing old log files.

logrotate runs daily via cron/systemd timer. It reads configuration from /etc/logrotate...

Options & Flags

`-d`	Debug mode (dry run)
`-f`	Force rotation
`-v`	Verbose output
`-s`	Use alternate state file

Practical Examples

Example: Force rotation

```
$ sudo logrotate -f /etc/logrotate.d/nginx
```

Forces immediate rotation of nginx logs.

Example: Test configuration

```
$ logrotate -d /etc/logrotate.d/myapp
```

Shows what would happen without actually rotating.

Example: Custom config example

```
$ /var/log/myapp/*.log {\n    daily\n    rotate 30\n    compress\n    delaycompress\n    missingok\n    notifempty\n    create 0644
```

Rotates daily, keeps 30 days, compresses, and reloads the service.

Example: Verbose run

```
$ sudo logrotate -v /etc/logrotate.conf
```

Runs logrotate with verbose output showing all actions.

Example: Check status

```
$ cat /var/lib/logrotate/status
```

Shows when each log was last rotated.

Tips & Best Practices

Pro Tip: Always test with -d: logrotate -d config tests your configuration without making changes. Always verify before deploying.

Note: Configuration directives: daily/weekly/monthly for frequency. rotate N keeps N old files. compress uses gzip. missingok ignores missing files.

Warning: postrotate scripts: Many services need HUP/reload after rotation: postrotate systemctl reload nginx endscript. Without this, the service writes to the old (now renamed) file.

$ watch

Beginner

Execute a program periodically and show output

watch executes a command repeatedly at a specified interval, displaying the output full-screen. It is perfect for monitoring changes in real time - disk usage, network connections, process lists, and any command output.

watch clears the screen and re-runs the command, showing the current time, i...

Options & Flags

`-n N`	Update every N seconds (default 2)
`-d`	Highlight differences between updates
`-t`	Hide the header
`-g`	Exit when output changes
`-e`	Exit on error

Practical Examples

Example: Monitor disk space

```
$ watch -n 10 df -h
```

Updates disk usage display every 10 seconds.

Example: Monitor connections

```
$ watch -n 1 "ss -s"
```

Shows network connection summary every second.

Example: Highlight changes

```
$ watch -d "ls -la /var/log/"
```

Highlights file changes (new files, size changes).

Example: Monitor process

```
$ watch -n 1 "ps aux | grep nginx | grep -v grep"
```

Monitors nginx processes every second.

Example: Wait for file

```
$ watch -g "ls /tmp/results/"
```

Exits as soon as a file appears in the directory.

Tips & Best Practices

Pro Tip: Quote complex commands: Use quotes for pipes: watch "command | grep filter". Without quotes, the pipe is interpreted by the current shell.

Note: -d highlights changes: -d highlights what changed between refreshes. Very useful for spotting changes in large outputs.

Warning: Not all commands work: Commands using colors or terminal features may display poorly. Use --color flag or pipe through col -b if needed.