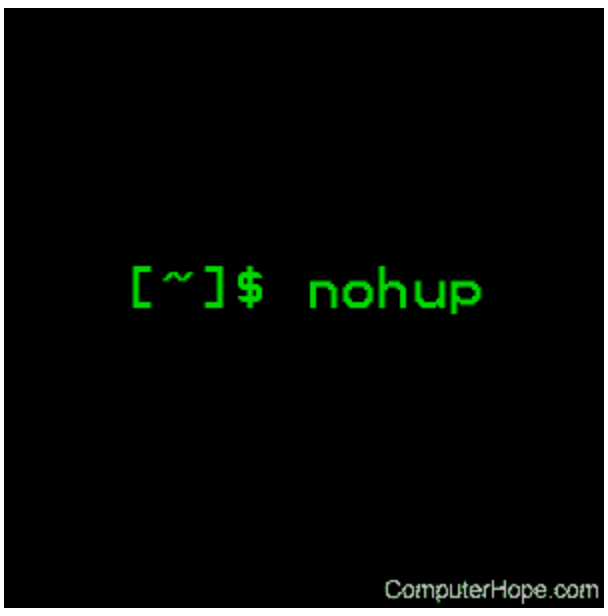
# Linux nohup command

Updated: 03/13/2021 by Computer Hope

On Unix-like operating systems, the **nohup** command executes another command, and instructs the system to continue running it even if the session is disconnected.




This page covers the GNU/Linux version of **nohup**.

**Page contents**



## Description

When using the command shell, prefixing a command with **nohup** prevents the command from being aborted automatically when you log out or exit the shell.

The name **nohup** stands for "no hangup." The hangup (**HUP**) signal, which is normally sent to a process to inform it the user has logged off (or "hung up"), is intercepted by **nohup**, allowing the process to continue running.

# Syntax

```
nohup command [command-argument ...]
```

```
nohup --help | --version
```

## Options

| | |
|---|---|
| **--help** | Display a help message and exit. |
| **--version** | Output version information and exit. |

## Notes

If standard input is a terminal, **nohup** redirects it from **/dev/null**. Therefore, terminal input is not possible when running a command with **nohup**.

If standard output is a terminal, command output is appended to the file **nohup.out** if possible, or **$HOME/nohup.out** otherwise.

If standard error is a terminal, it is redirected to standard output.

To save output to a file named *file*, use "**nohup** *command* **>** *file*".

# Examples

```
nohup mycommand
```

Run the command **mycommand**. It does not receive input. All output, including any error messages, is written to the file **nohup.out** in the working directory, or in your home directory. If **mycommand** is running when you log out or close the terminal, **mycommand** does not terminate.

```
nohup mycommand &
```

Same as the previous command, but this form (when using the bash shell) returns you immediately to the shell prompt. The "**&**" symbol at the end of the command instructs bash to run **nohup mycommand** in the

background. It can be brought back to the foreground with the **fg** bash builtin command.

When using **&**, you see the bash job ID in brackets, and the PID (process ID) listed after. For example:

```
[1] 25132
```

You can use the PID to terminate the process prematurely. For instance, to send it the **TERM** (terminate) signal with the **kill** command:

```
kill -9 25132
```

## Related commands

**nice** — Invoke a command with an altered scheduling priority.