

Job Control Cheatsheet

By Dejan Panovski • Updated on Oct 16, 2026

Quick reference for Bash job control: background jobs, jobs, fg, bg, wait, and keeping processes alive after logout with disown, nohup, and setsid

Job control lets you run commands in the background, suspend and resume them, and move them between the foreground and background of the current shell. This cheatsheet covers `&`, `jobs`, `fg`, `bg`, `kill`, `wait`, `disown`, and `nohup` as they work in Bash.

Run in the Background

End a command with `&` to get the prompt back right away.

<code>command &</code>	Run a command in the background
<code>command > out.log 2>&1 &</code>	Background job with output saved to a file
<code>command > /dev/null 2>&1 &</code>	Background job with all output discarded
<code>(cmd1; cmd2) &</code>	Run a command group as one job
<code>cmd1 & cmd2 &</code>	Start two jobs in parallel
<code>echo \$!</code>	Print the PID of the last background job

List Jobs

`jobs` shows only jobs started from the current shell. Use `ps` for processes from other sessions.

<code>jobs</code>	List jobs with their state
<code>jobs -l</code>	Include process IDs
<code>jobs -p</code>	Print only process group leader PIDs
<code>jobs -r</code>	List running jobs only
<code>jobs -s</code>	List stopped jobs only
<code>jobs -n</code>	List jobs whose state changed since the last notice
<code>jobs %2</code>	Show job 2 only

Job Specifications

`fg` , `bg` , `kill` , `wait` , and `disown` accept these job references.

<code>%1</code>	Job number 1
<code>%%</code> or <code>%+</code>	Current job, marked + in jobs
<code>%-</code>	Previous job, marked - in jobs
<code>%ping</code>	Job whose command starts with ping
<code>%?log</code>	Job whose command contains log

Without a jobspec, `fg` , `bg` , and `disown` act on the current job.

Foreground and Background

Suspend a running command, then decide where it continues.

<code>Ctrl+Z</code>	Suspend the foreground job (SIGTSTP)
<code>bg</code>	Resume the current job in the background
<code>bg %2</code>	Resume job 2 in the background
<code>fg</code>	Bring the current job to the foreground
<code>fg %1</code>	Bring job 1 to the foreground
<code>%1</code>	Short form of <code>fg %1</code>
<code>%1 &</code>	Short form of <code>bg %1</code>

Stop and Kill Jobs

`kill` accepts a jobspec as well as a PID.

<code>Ctrl+C</code>	Interrupt the foreground job (SIGINT)
<code>Ctrl+\</code>	Quit the foreground job (SIGQUIT)
<code><u>kill %1</u></code>	Send SIGTERM to job 1
<code>kill -9 %1</code>	Force kill job 1 (SIGKILL)
<code>kill -STOP %1</code>	Pause job 1; the signal cannot be caught
<code>kill -CONT %1</code>	Continue a stopped job in the background
<code>kill \$(jobs -p)</code>	Send SIGTERM to every job in the current shell

Wait for Jobs

Use `wait` in scripts to block until background work finishes.

<code>wait</code>	Wait for all background jobs
<code>wait %1</code>	Wait for job 1
<code>wait "\$pid"; echo \$?</code>	Wait for a PID and print its exit status
<code>wait -n</code>	Wait for the next job to finish
<code>wait -n -p id</code>	Store the PID of the finished job in <code>id</code> (Bash 5.1+)
<code>for h in web1 web2; do ssh "\$h" uptime & done; wait</code>	Run commands in parallel, then wait for all

Keep Jobs Running After Logout

Redirect output before you log out. A job that keeps writing to a closed terminal can fail.

<code>disown %1</code>	Remove job 1 from the job table
<code>disown -h %1</code>	Keep job 1 listed but do not send it SIGHUP on exit
<code>disown -a</code>	Remove all jobs
<code>disown -r</code>	Remove running jobs only
<code>Ctrl+Z, bg, disown -h</code>	Move a running foreground command to the background and protect it
<code>nohup command &</code>	Ignore SIGHUP; output goes to <code>nohup.out</code>
<code>nohup command > out.log 2>&1 &</code>	Ignore SIGHUP and write output to a named log
<code>setsid -f command > out.log 2>&1 < /dev/null</code>	Start in a new session, detached from the terminal

Job States

States shown by `jobs` and the shell's job notices.

Running	Job runs in the background
Stopped	Job is suspended and waits for <code>fg</code> or <code>bg</code>
Stopped (tty input)	Background job tried to read from the terminal; use <code>fg</code>
Done	Job finished with exit status 0
Exit 3	Job finished with a non-zero exit status
Terminated	Job ended on SIGTERM
Killed	Job ended on SIGKILL

Shell Options

Settings that change how Bash handles jobs.

<code>set -m</code>	Enable job control in a script (on by default in interactive shells)
<code>set -b</code>	Report finished jobs immediately instead of at the next prompt
<code>shopt -s checkjobs</code>	List running and stopped jobs before the shell exits
<code>shopt -s huponexit</code>	Send SIGHUP to all jobs when an interactive login shell exits
<code>stty tostop</code>	Stop background jobs that write to the terminal
<code>stty -tostop</code>	Let background jobs write to the terminal (default)

Related Guides

Use these guides for the longer explanations behind these commands.

How to Run Linux Commands in the Background	<code>&</code> , <code>disown</code> , <code>nohup</code> , and <code>setsid</code> in detail
jobs Command in Linux	Job specifications, states, and options
nohup Command in Linux	<code>nohup.out</code> , redirection, and logout behavior
How to Use the kill Command in Linux	Signals and how to send them
Bash wait Command	Exit status handling for background jobs
tmux Cheatsheet	Sessions you can detach and reattach later
screen Cheatsheet	The same idea with GNU Screen