NAME

launchctl — Interfaces with launchd

SYNOPSIS

launchctl subcommand [arguments ...]

DESCRIPTION

launchctl interfaces with **launchd** to manage and inspect daemons, agents and XPC services.

SUBCOMMANDS

launchctl allows for detailed examination of **launchd**'s data structures. The fundamental structures are domains, services, and endpoints. A domain manages the execution policy for a collection of services. A service may be thought of as a virtual process that is always available to be spawned in response to demand. Each service has a collection of endpoints, and sending a message to one of those endpoints will cause the service to launch on demand. Domains advertise these endpoints in a shared namespace and may be thought of as synonymous with Mach bootstrap subsets.

Many subcommands in **launchctl** take a specifier which indicates the target domain or service for the subcommand. This specifier may take one of the following forms:

system/[service-name]
: Targets the system domain or a service within the system domain. The system domain manages the root Mach bootstrap and is considered a privileged execution context. Anyone may read or query the system domain, but root privileges are required to make modifications.

user/<uid>/[service-name]
: Targets the user domain for the given UID or a service within that domain. A user domain may exist independently of a logged-in user. User domains do not exist on iOS.

login/<asid>/[service-name]
: Targets a user-login domain or service within that domain. A user-login domain is created when the user logs in at the GUI and is identified by the audit session identifier associated with that login. If a user domain has an associated login domain, the **print** subcommand will display the ASID of that login domain. User-login domains do not exist on iOS.

gui/<uid>/[service-name]
: Another form of the **login** specifier. Rather than specifying a user-login domain by its ASID, this specifier targets the domain based on which user it is associated with and is generally more convenient.

  Note: GUI domains and user domains share many resources. For the purposes of the Mach bootstrap name lookups, they are "flat", so they share the same set of registered names. But they still have discrete sets of services. So when printing the user domain's contents, you may see many Mach bootstrap name registrations from services that exist in the GUI domain for that user, but you will not see the services themselves in that list.

pid/<pid>/[service-name]
: Targets the domain for the given PID or a service within that domain. Each process on the system will have a PID domain associated with it that consists of the XPC services visible to that process which can be reached with xpc_connection_create(3).

For instance, when referring to a service with the identifier **com.apple.example** loaded into the GUI domain of a user with UID 501, **domain-target** is **gui/501/**, **service-name** is **com.apple.example**, and **service-target** is **gui/501/com.apple.example**.

SUBCOMMANDS

bootstrap | bootout domain-target [service-path service-path2 ...] | service-target

Bootstraps or removes domains and services. When service arguments are present, bootstraps and correspondingly removes their definitions into the domain. Services may be specified as a series of paths or a service identifier. Paths may point to XPC service bundles, launchd.plist(5) s, or a directories containing a collection of either. If there were one or more errors while bootstrapping or removing a collection of services, the problematic paths will be printed with the errors that occurred.

If no paths or service target are specified, these commands can either bootstrap or remove a domain specified as a domain target. Some domains will implicitly bootstrap pre-defined paths as part of their creation.

enable | disable service-target
Enables or disables the service in the requested domain. Once a service is disabled, it cannot be loaded in the specified domain until it is once again enabled. This state persists across boots of the device. This subcommand may only target services within the system domain or user and user-login domains.

kickstart [**-kp**] service-target
Instructs **launchd** to run the specified service immediately, regardless of its configured launch conditions.

-k If the service is already running, kill the running instance before restarting the service.

-p Upon success, print the PID of the new process or the already-running process to stdout.

attach [**-ksx**] service-target
Attaches the system's debugger to the process currently backing the specified service. By default, if the service is not running, this subcommand will block until the service starts.

-k If the service is already running, kill the running instance.

-s Force the service to start.

-x Attach to xpcproxy(3) before it execs and becomes the service process. This flag is generally not useful for anyone but the **launchd** maintainer.

debug service-target [--program <program path>] [--guard-malloc] [--malloc-stack-logging] [--debug-libraries] [--introspection-libraries] [--NSZombie] [--32] [--stdin] [--stdout] [--stderr] [--environment] [--] [argv0 argv1 argv2 ...]
Configures the next invocation of a service for debugging. This subcommand allows you to temporarily replace the main executable of the service with one at a different path, enable libgmalloc(3), set environment variables, set the argument vector and more. This is a convenient alternative to editing the launchd.plist(5) for the service and then reloading it, as the additional debugging properties are cleared once the service has run once with them.

--program <program-path>
Instructs launchd(8) to use program-path as the service's executable.

--guard-malloc
Turns on libgmalloc(3) for the service.

--malloc-stack-logging
Turns on malloc(3) stack logging for the service.

--malloc-nano-allocator
Turns on the malloc(3) nano allocator for the service.

--debug-libraries

Sets the DYLD_IMAGE_SUFFIX for the service to "_debug", which prefers the debug variants of libraries if they exist. See dyld(1) for more information.

--introspection-libraries
Adds /usr/lib/system/introspection to the DYLD_LIBRARY_PATH environment variable for the service. This causes the system to prefer the introspection variants of libraries if they exist.

--NSZombie
Enables NSZombie.

--32 Runs the service in the appropriate 32-bit architecture. Only available on 64-bit platforms.

--stdin [stdin-path]
Sets the service's standard input to be stdin-path. If no file is given, uses the current terminal as the service's standard input. If stdin-path does not exist, it is created.

--stdout [stdout-path]
Sets the service's standard output to be stdout-path. If no file is given, uses the current terminal as the service's standard output. If stdout-path does not exist, it is created.

--stderr [stderr-path]
Sets the service's standard error to be stderr-path. If no file is given, uses the current terminal as the service's standard error. If stderr-path does not exist, it is created.

--environment VARIABLE0=value VARIABLE1=value ...
Sets the given environment variables on the service.

-- argv0 argv1 ...
Any arguments following the -- are given to the service as its argument vector.

IMPORTANT: These arguments replace the service's default argument vector; they are not merged in any way. The first argument following -- is given as the initial (zeroth) element of the service's argument vector. As with the ProgramArguments launchd.plist(5) key, you should read carefully and understand the execve(2) man page.

kill signal-name | signal-number service-target
Sends the specified signal to the specified service if it is running. The signal number or name (SIGTERM, SIGKILL, etc.) may be specified.

blame service-target
If the service is running, prints a human-readable string describing why **launchd** launched the service. Note that services may run for many reasons; this subcommand will only show the most proximate reason. So if a service was run due to a timer firing, this subcommand will print that reason, irrespective of whether there were messages waiting on the service's various endpoints. This subcommand is only intended for debugging and profiling use and its output should not be relied upon in production scenarios.

print domain-target | service-target
Prints information about the specified service or domain. Domain output includes various properties about the domain as well as a list of services and endpoints in the domain with state pertaining to each. Service output includes various properties of the service, including information about its origin on-disk, its current state, execution context, and last exit status.

IMPORTANT: This output is NOT API in any sense at all. Do NOT

rely on the structure or information emitted for ANY reason. It may change from release to release without warning.

print-cache
: Prints the contents of the **launchd** service cache.

print-disabled domain-target
: Prints the list of disabled services in the specified domain.

plist [segment,section] Mach-O
: Prints the the property list embedded in the __TEXT,__info_plist segment/section of the target Mach-O or the specified segment/section.

procinfo pid
: Prints information about the execution context of the specified PID. This information includes Mach task-special ports and exception ports (and when run against a DEVELOPMENT launchd, what names the ports are advertised as in the Mach bootstrap namespace, if they are known to **launchd**) and audit session context. This subcommand is intended for diagnostic purposes only, and its output should not be relied upon in production scenarios. This command requires root privileges.

hostinfo
: Prints information about the system's host-special ports, including the host-exception port. This subcommand requires root privileges.

resolveport owner-pid port-name
: Given a PID and the name of a Mach port right in that process' port namespace, resolves that port to an endpoint name known to **launchd**. This subcommand requires root privileges.

examine [tool arg0 arg1 @PID ...]
: Causes **launchd** to fork(2) itself for examination by a profiling tool and prints the PID of this new instance to stdout. You are responsible for killing this snapshot when it is no longer needed.

  Many profiling tools cannot safely examine **launchd** because they depend on the functionality it provides. This subcommand creates an effective snapshot of **launchd** that can be examined independently. Note that on Darwin platforms, fork(2) is implemented such that only the thread which called fork(2) is replicated into the new child process, so this subcommand is not useful for examining any thread other than the main event loop.

  This subcommand takes an optional invocation of a tool to be used on the **launchd** snapshot. Where you would normally give the PID of the process to be examined in the tool's invocation, instead specify the argument "@PID", and **launchctl** will substitute that argument with the PID of the launchd snapshot in its subsequent execution of the tool. If used in this form, **launchctl** will automatically kill the snapshot instance when the examination tool exits.

  This subcommand may only be used against a DEVELOPMENT **launchd**.

config system | user parameter value
: Sets persistent configuration information for launchd(8) domains. Only the system domain and user domains may be configured. The location of the persistent storage is an implementation detail, and changes to that storage should only be made through this subcommand. A reboot is required for changes made through this subcommand to take effect.

  Supported configuration parameters are:

  umask
  : Sets the umask(2) for services within the target domain to the value specified by value. Note that this value is parsed by strtoul(3) as an octal-encoded number, so there is no need to prefix it with a leading '0'.

path Sets the PATH environment variable for all services within the target domain to the string value. The string value should conform to the format outlined for the PATH environment variable in environ(7). Note that if a service specifies its own PATH, the service-specific environment variable will take precedence.

NOTE: This facility cannot be used to set general environment variables for all services within the domain. It is intentionally scoped to the PATH environment variable and nothing else for security reasons.

reboot [system|userspace|halt|logout|apps]
Instructs **launchd** to begin tearing down userspace. With no argument given or with the system argument given, **launchd** will make the reboot(2) system call when userspace has been completely torn down. With the halt argument given, **launchd** will make the reboot(2) system call when userspace has been completely torn down and pass the RB_HALT flag, halting the system and not initiating a reboot.

With the userspace argument given, **launchd** will re-exec itself when userspace has been torn down and bring userspace back up. This is useful for rebooting the system quickly under conditions where kernel data structures or hardware do not need to be re-initialized.

With the logout argument given, launchd will tear down the caller's GUI login session in a manner similar to a logout initiated from the Apple menu. The key difference is that a logout initiated through this subcommand will be much faster since it will not give apps a chance to display modal dialogs to block logout indefinitely; therefore there is data corruption risk to using this option. Only use it when you know you have no unsaved data in your running apps.

With the apps argument given, launchd will terminate all apps running in the caller's GUI login session that did not come from a launchd.plist(5) on-disk. Apps like Finder, Dock and SystemUIServer will be unaffected. Apps are terminated in the same manner as the logout argument, and all the same caveats apply.

error [posix|mach|bootstrap] code
Prints a human-readable string of the given error code. By default, **launchctl** will attempt to guess which error domain the code given belongs to. The caller may optionally specify which domain (either posix, mach, or bootstrap) to interpret the given code as an error from that subsystem.

variant Prints the **launchd** variant currently active on the system. Possible variants include RELEASE, DEVELOPMENT and DEBUG.

version Prints the **launchd** version string.

LEGACY SUBCOMMANDS

Legacy subcommands select the target domain based on whether they are executed as root or not. When executed as root, they target the system domain.

load | unload [**-wF**] [**-S** sessiontype] [**-D** searchpath] paths ...
Recommended alternative subcommands: bootstrap | bootout | enable | disable

Load the specified configuration files or directories of configuration files. Jobs that are not on-demand will be started as soon as possible. All specified jobs will be loaded before any of them are allowed to start. Note that per-user configuration files (LaunchAgents) must be owned by root (if they are located in **/Library/LaunchAgents**) or the user loading them (if they are located in **$HOME/Library/LaunchAgents**). All system-wide daemons (LaunchDaemons) must be owned by root. Configuration files must disallow group and world writes. These

restrictions are in place for security reasons, as allowing writability to a launchd configuration file allows one to specify which executable will be launched.

Note that allowing non-root write access to the **/System/Library/LaunchDaemons** directory WILL render your system unbootable.

-w Overrides the Disabled key and sets it to false or true for the load and unload subcommands respectively. In previous versions, this option would modify the configuration file. Now the state of the Disabled key is stored elsewhere on- disk in a location that may not be directly manipulated by any process other than **launchd.**

-F Force the loading or unloading of the plist. Ignore the Disabled key.

-S sessiontype
Some jobs only make sense in certain contexts. This flag instructs **launchctl** to look for jobs in a different location when using the -D flag, and allows **launchctl** to restrict which jobs are loaded into which session types. Sessions are only relevant for per-user **launchd** contexts. Relevant sessions are Aqua (the default), Background and LoginWindow. Background agents may be loaded independently of a GUI login. Aqua agents are loaded only when a user has logged in at the GUI. LoginWindow agents are loaded when the LoginWindow UI is displaying and currently run as root.

-D searchpath
Load or unload all plist(5) files in the search path given. This option may be thought of as expanding into many individual paths depending on the search path given. Valid search paths include "system," "local," and "all." When providing a session type, an additional search path is available for use called "user." For example, without a session type given, "-D system" would load from or unload all property list files from **/System/Library/LaunchDaemons.** With a session type passed, it would load from **/System/Library/LaunchAgents.** Note that **launchctl** no longer respects the network search path.

In a previous version of launchd, these search paths were called "domains", hence -D. The word "domain" is now used for a totally different concept.

NOTE: Due to bugs in the previous implementation and long-standing client expectations around those bugs, the load and unload subcommands will only return a non-zero exit code due to improper usage. Otherwise, zero is always returned.

submit **-l** label [**-p** executable] [**-o** stdout-path] [**-e** stderr-path] -- command [arg0] [arg1] [...]
A simple way of submitting a program to run without a configuration file. This mechanism also tells launchd to keep the program alive in the event of failure.

-l label
What unique label to assign this job to launchd.

-p program
What program to really execute, regardless of what follows the -- in the submit sub-command.

-o stdout-path
Where to send the stdout of the program.

-e stderr-path
Where to send the stderr of the program.

remove label
: Remove the job from launchd by label. This subcommand will return immediately and not block until the job has been stopped.

start label
: Start the specified job by label. The expected use of this subcommand is for debugging and testing so that one can manually kick-start an on-demand server.

stop label
: Stop the specified job by label. If a job is on-demand, launchd may immediately restart the job if launchd finds any criteria that is satisfied.

list [-x] [label]
: Recommended alternative subcommand: print

  With no arguments, list all of the jobs loaded into **launchd** in three columns. The first column displays the PID of the job if it is running. The second column displays the last exit status of the job. If the number in this column is negative, it represents the negative of the signal which stopped the job. Thus, "-15" would indicate that the job was terminated with SIGTERM. The third column is the job's label. If [label] is specified, prints information about the requested job.

  -x This flag is no longer supported.

setenv key value
: Specify an environment variable to be set on all future processes launched by **launchd** in the caller's context.

unsetenv key
: Specify that an environment variable no longer be set on any future processes launched by **launchd** in the caller's context.

getenv key
: Print the value of an environment variable that **launchd** would set for all processes launched into the caller's context.

export
: Export all of the environment variables of **launchd** for use in a shell eval statement.

getrusage self | children
: Get the resource utilization statistics for **launchd** or the children of **launchd**. This subcommand is not implemented.

limit [cpu | filesize | data | stack | core | rss | memlock | maxproc | maxfiles] [both [soft | hard]]
: With no arguments, this command prints all the resource limits of **launchd** as found via getrlimit(2). When a given resource is specified, it prints the limits for that resource. With a third argument, it sets both the hard and soft limits to that value. With four arguments, the third and forth argument represent the soft and hard limits respectively. See setrlimit(2).

shutdown
: Tell **launchd** to prepare for shutdown by removing all jobs. This subcommand is not implemented.

umask [newmask]
: Get or optionally set the umask(2) of **launchd**. This subcommand is not implemented.

bslist [PID | ..] [-j]
: This subcommand is not implemented and has been superseded by the print subcommand, which provides much richer information.

bsexec PID command [args]
: This executes the given command in as similar an execution context as possible to the target PID. Adopted attributes include the Mach bootstrap namespace, exception server and security audit session. It does not modify the process' credentials (UID, GID, etc.) or adopt any environment variables

from the target process. It affects only the Mach bootstrap context and directly-related attributes.

asuser UID command [args]
This executes the given command in as similar an execution context as possible to that of the target user's bootstrap. Adopted attributes include the Mach bootstrap namespace, exception server and security audit session. It does not modify the process' credentials (UID, GID, etc.) or adopt any user-specific environment variables. It affects only the Mach bootstrap context and directly- related attributes.

bstree This subcommand is not implemented and has been superseded by the print subcommand, which provides much richer information.

managerpid
This prints the PID of the launchd which manages the current bootstrap. In prior implementations, there could be multiple **launchd** processes each managing their own Mach bootstrap subsets. In the current implementation, all bootstraps are managed by one process, so this subcommand will always print "1".

manageruid
This prints the UID associated with the caller's launchd context.

managername
This prints the name of the launchd job manager which manages the current launchd context. See LimitLoadToSessionType in launchd.plist(5) for more details.

help Print out a quick usage statement.

CAVEATS

The output produced by the "legacy" subcommands (chiefly list) should match their output on previous OS X releases. However, the output of newer subcommands does not conform to any particular format and is not guaranteed to remain stable across releases. These commands are intended for use by human developers and system administrators, not for automation by programs or scripts. Their output does not constitute an API and no promises of forward compatibility are offered to programs that attempt to parse it.

DEPRECATED AND REMOVED FUNCTIONALITY

launchctl no longer has an interactive mode, nor does it accept commands from stdin. The **/etc/launchd.conf** file is no longer consulted for subcommands to run during early boot time; this functionality was removed for security considerations. While it was documented that **$HOME/.launchd.conf** would be consulted prior to setting up a user's session, this functionality was never implemented.

launchd no longer uses Unix domain sockets for communication, so the **LAUNCHD_SOCKET** environment variable is no longer relevant and is not set.

launchd no longer loads configuration files from the network

FILES

~/Library/LaunchAgents	Per-user agents provided by the user.
/Library/LaunchAgents	Per-user agents provided by the administrator.
/Library/LaunchDaemons	System wide daemons provided by the administrator.
/System/Library/LaunchAgents	OS X Per-user agents.
/System/Library/LaunchDaemons	OS X System wide daemons.

EXIT STATUS

launchctl will exit with status 0 if the subcommand succeeded. Otherwise, it will exit with an error code that can be given to the error subcommand to be decoded into human-readable form.

SEE ALSO

launchd.plist(5), launchd(8), audit(8), setaudit_addr(2)