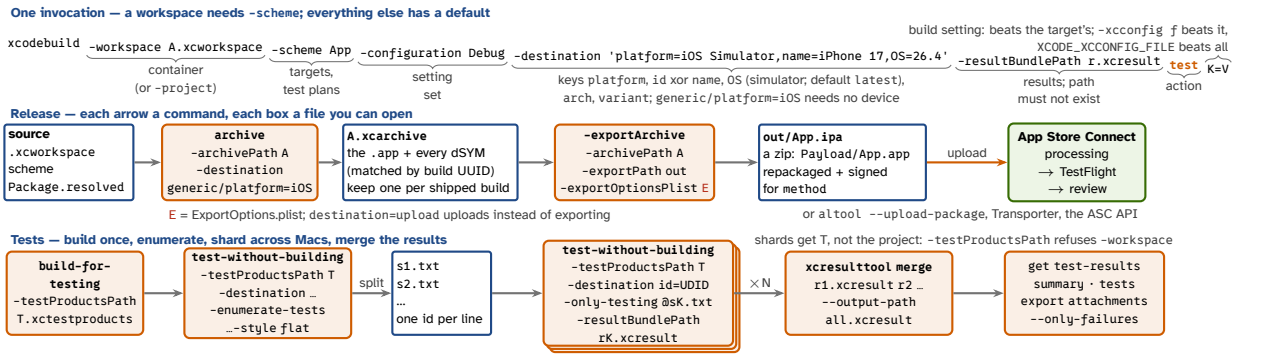
xcodebuild, simctl & friends — Xcode from the command line

In one line: `xcodebuild` runs one **action** of a **scheme** against a **destination** and leaves files you can open — an `.xcarchive`, an `.ipa`, an `.xcresult`. `simctl` drives simulators, `devicectl` phones; `xcode-select` / `DEVELOPER_DIR` pick **which Xcode** answers.



Build once, test anywhere

`build-for-testing` writes an `.xctestrun`: an editable plist of bundle paths, environment and arguments (Xcode 26+: even *disabled* plan configurations). Any Mac runs it with `test-without-building` — no source, no compiler. Tests are paths, `-only-testing Target/Class/method`; `@file` reads one a line; *only* beats *skip*.

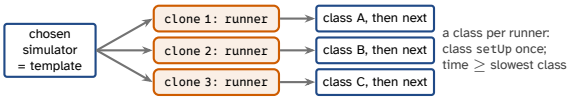
Two repetition loops that point opposite ways

`-retry-tests-on-failure` 🔴🔴🟢 stops at the first **pass** (≤3): a flake ends **green**

`-run-tests-until-failure` 🟢🟢🟢🟢🟢 ··· 🔴 stops at the first **fail** (≤100): caught

`-test-iterations N`: N runs, or either loop's cap.
`-test-repetition-relaunch-enabled YES`: a fresh process per run.

Parallel: the simulator you pick never runs a test



Several `-destinations` each run the *whole* suite; with parallel testing on, `-parallelize-tests-among-destinations` splits it.

One prefix, stripped on the way in

`TEST_RUNNER_Foo=Bar xcodebuild test` → each test runner: `Foo=Bar`

`SIMCTL_CHILD_Foo=Bar simctl launch booted id` → the app: `Foo=Bar`

`simctl` has no environment flag; the prefix *is* the interface. At `boot` it reaches the whole device — in a 2019 experiment, enough to give an iPhone 8 simulator Face ID.

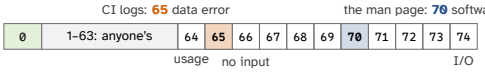
The `.xcresult`: query it, never parse it

`xcresulttool` speaks JSON; `help <command>` prints the schema.

What changed, Xcode 13 → 27

- **13** test repetition (the two loops); API-key signing on headless Macs
- **15** Xcode Server removed; `-enumerate-tests`; API-key uploads; `devicectl`
- **16** `get test-results`; an unmatched `-destination` no longer silently runs on the first compatible one; `-downloadPlatform -exportPath`; `devicectl --console`
- **26** Swift 6.2; explicit modules on (`SWIFT_ENABLE_EXPLICIT_MODULES=NO`); opt-in local compilation cache (`COMPILATION_CACHE_ENABLE_CACHING=YES`)
- **26.4** Swift 6.3; runtime exports gain `.exportBundle`; `runtime dyld_shared_cache update`
- **27** Swift 6.4; Apple-silicon Macs, macOS 26.6+; `-ld_classic` gone; storyboards compile with no simulator; `simctl/devicectl reboot`; `devicectl --json-output` to stdout

Exit codes: a 1980 mail header

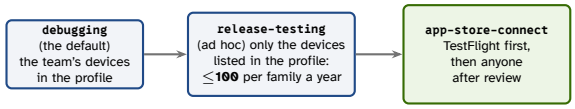


`xcodebuild` exits by sysexits(3): `sysexits.h`, first shipped in 4BSD (1980), written for delivermail — sendmail's predecessor — and the Berkeley network, starting at 64 "to reduce the possibility of clashing". Logs after `** BUILD/TEST FAILED **` show 65, an unmatched `-destination` 70: branch on non-zero, never on a number.

Traps

- ! `xcodebuild … | xcbeautify` without `set -o pipefail`: the pipeline returns the formatter's status — a red build reports green.
- ! No `OS=` is `latest`: a new runner image moves the iOS version. Pin `OS=` or `id=`.
- ! Xcode 27: `XCTAssert` inside a Swift Testing test (or `#expect` in XCTest) only warns, unless the test plan's interoperability setting says otherwise.

Export: `method` answers "who may install this?"



off the ladder: `enterprise` (your staff, no store) · `developer-id` (a Mac app outside the store); `development`, `ad-hoc`, `app-store` are deprecated aliases

The 100 is per product family per membership year; disabling a device does not give the slot back, only renewal does. `manageAppVersionAndBuildNumber`: on upload Xcode raises a used build number and syncs the extensions', in a staged copy, never the archive — off when CI owns it. `testFlightInternalTestingOnly`: internal TestFlight only. Manual signing: `provisioningProfiles` {bundle id → profile}.

Signing with nobody at the keyboard

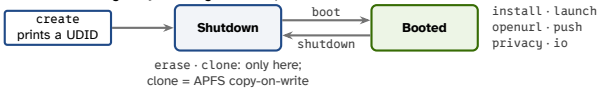
`-allowProvisioningUpdates` lets `xcodebuild` make certificates and fetch profiles itself; an App Store Connect API key (key path, key ID, issuer ID) stands in for a logged-in account and also uploads.

Swift packages, plugins, macros on CI

- `Package.resolved` is the lockfile: commit it; `-disableAutomaticPackageResolution` makes CI build exactly those versions.
- Checkouts land in `DerivedData/SourcePackages`; `-clonedSourcePackagesDirPath P` moves them somewhere a cache survives.
- `-scmProvider system`: the Mac's own git, its config and SSH keys.
- Nobody is there to click "Trust & Enable": `-skipPackagePluginValidation`, `-skipMacroValidation` — in the tool's words, "can be a security risk".

A simulator is a userspace, not a virtual machine

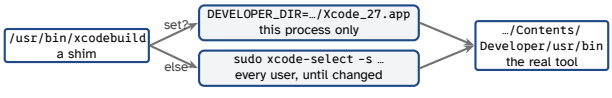
The iOS userspace — its own `launchd`, daemons and frameworks — runs on the *Mac*'s kernel, sharing its filesystem with a home of its own. CPU and memory limits are not simulated: on a Mac with a terabyte of RAM the "iPhone" can allocate a terabyte. Simulator timings say nothing about a phone.



`erase` · `clone`: only here; clone = APFS copy-on-write

What a phone will not let a script do, a simulator will: `privacy … grant` (no alert to tap), `push` an `.apns` file, `keychain … add-root-cert` (trust your proxy), `status_bar … override --time 9:41`. Why 9:41: Apple designed keynotes so the reveal lands about 40 minutes in and the screen matches the audience's watches — the 2007 iPhone showed 9:42, the 2010 iPad 9:41. Script with UDIDs; `booted` = the one booted device. Phones: `devicectl` (CoreDevice, iOS 17+), `--device` by name, UDID or ECID; parse `--json-output`, not text.

Which Xcode answers



`xcrun --find T` resolves without running; `xcrun --sdk S` exports SDKROOT to the tools it starts, but not to `xcodebuild` or `make`. `xcodes` + `.xcode-version` pin the Xcode. Fresh Mac: `-runFirstLaunch`; `-downloadPlatform iOS -exportPath D` once, `-importPlatform` on every Mac.

Xcode Cloud

Runs only `ci_post_clone.sh`, `ci_pre_xcodebuild.sh` and `ci_post_xcodebuild.sh` from `ci_scripts/` beside the project — the last *even if xcodebuild failed*. No shebang → `zsh file`; no `sudo`; created files are deleted; secrets print as asterisks. 25 compute hours a month are included.

Related: `ci-pipeline-design` · `fastlane-deep` · `github-actions-deep` · Code signing & iOS CI/CD · `macos-keychain-security-cli` · Flaky tests · CI · coverage · TDD · `xctest-and-swift-testing` · Crashes & symbolication · Modularization & Swift Package Manager · `macos-power-user-cli`