

Why use CTL and Ant?

Ant is useful for automating all kinds of software build processes across multiple OS platforms: compiling, invoking unit tests, packaging applications, etc. Naturally, after building applications with Ant, one would like to extend that process to stage, deploy and even manage application operations using Ant. Better yet, it would be nice if the same set of Ant build files used to manage the lifecycle of applications in development could be used throughout other environments like QA and production.

However, writing deployment and operations scripts in Ant quickly gets tricky and you are also faced with new requirements like the ability to package up scripts in such a way that they can be migrated and handed off to be used by people in different groups like QA and operations.

Here are some issues that Ant users typically face when extending their use of Ant to automate application deployment and operations:

- **File distribution:** After the build is complete, build artifacts must be distributed to their target hosts. This can be done with the [Scp](#) task but the process doesn't stop there. Often times files will need extraction, filtering, renaming and these steps may need to be done in between server restarts. [Go to # Cookbook: file distribution](#)
- **Process management:** During the installation process one frequently needs to restart a process to begin using files from the new release. Doing so involves looking for a running process, sending kill signals, and running executables. [Go to # Cookbook: process management](#)
- **Remote execution:** Outside of build environments, applications tend to run across multiple machines. This means an Ant user must work out how to remotely invoke actions. Of course, the [Sshexec](#) task is an obvious place to start. One quickly finds that formulating a series of script fragments that get executed on the target becomes complicated and unwieldy. [Go to # Cookbook: remote execution](#)
- **Parallel execution:** Sometimes a large number of steps must be performed either because they must be done across many machines, or there are many individual steps. Other times, some procedures take a long time while others run quickly. These are all reasons to prefer a parallel execution mode. Ant does provide the [Parallel](#) task but it does not let you dynamically define what tasks should be executed in different environments. [Go to #](#)

[Cookbook: parallel execution](#)

- **Centralized coordination:** The more steps that must occur across multiple targets the more desperate the need for centralized coordination. This is one of the more challenging parts to building deployment procedures. There are low-level steps that are relatively simple. There are also high-level steps that govern a larger process. Managing these two levels independently is key to maintainable automation code. [Go to # Cookbook: coordination](#)
- **Flexible parameterization:** Ant users are accustomed to using build.properties files to generalize their Ant code. In an operations environment, command-line option setting is also expected. Ant users use the `-Dproperty=value` declarations but there isn't a formalized option handling capability that complement the build.properties files typically used. [Go to # Cookbook: options](#)
- **An interface for operations:** People in operations depend on clean and consistent interfaces. This means proper command line options, commands that have usage statements and supporting documentation. These are more than "nice to haves" for them because they must support a large number of applications and procedures (beyond your own even!). [Go to # Cookbook: runbooks](#)

Anyone who has attempted or is using Ant outside of the build arena already has faced many of these challenges. Others that may be contemplating the using Ant for deployment may wonder how they could possibly meet any of these challenges. CTL addresses each of these problem areas, and lets you do it using the Ant skills you already have. It's as easy as moving your deployment logic out of your build scripts and into some CTL Ant commands.

[Next: Ant in the 'hood #](#)