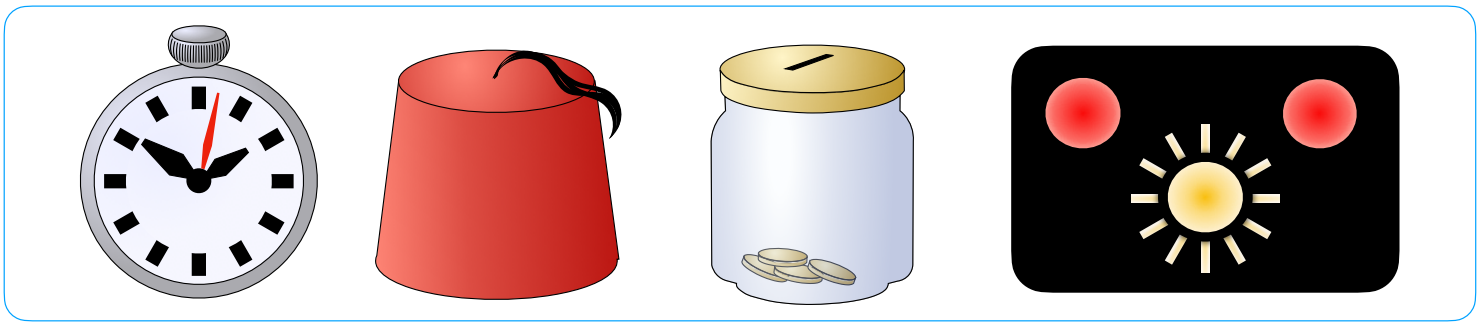
10 Tips for Continuous Integration



10 Tips for Continuous Integration

Continuous Integration is the antidote to "Integration Hell".
When we are developing software we want to avoid the complex, difficult and frustrating task of bringing diverse pieces of work together into a functioning system. We want to know that the code we are writing works with everyone else's – and we want to know this early and often.

CI is one of the **Extreme Programming** disciplines, first described in the C2 wiki (Kent Beck, et al) which asserts that code ownership and conflicts between people editing, are minimised when engineers don't hold onto a module for more than a moment. When correct changes are available to everyone, almost instantly, the system is always integrated and there is only one interesting version – the current one.

CI means little, or **no branching** (dividing up the work and working on separate pieces independently). Instead we make small changes to Trunk, or Master, and continuously evaluate them. If branches exist at all, they are tiny and short-lived, no longer than a day. Recent "DevOps Research & Assessment" reports found that merging frequently is a reliable predictor of higher Throughput, Stability and overall software performance.

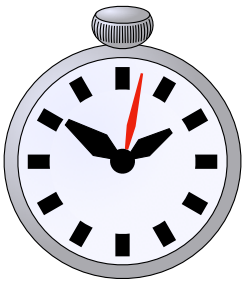
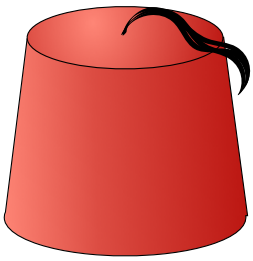
"Continuous" means at least very often. We should **Commit at least once per day** as a minimum, but probably more often than you think! We need to evaluate our changes as frequently as we can, and get feedback multiple times a day, so that we get several chances to fix any problems that same day.

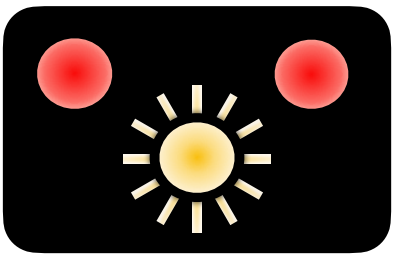
CI is an extremely important practice for **Continuous Delivery**. We want to maintain our software in a releasable state and keep the route to production open and moving. So we don't stall while we fix problems, nor will we leave bugs to be fixed later.

CI is not just a technical discipline. There are some great **CI tools** available to help, but the way we think about making a change, and the approach we take as individuals within a team, can have the biggest impact on our success. These techniques are an effective way to build collaborative teams.

My Top Tips

1. I work in small steps: Committing and testing small changes, not whole features. Small changes are less likely to hide complex problems, simpler to test, and easier to fix.
2. I run Commit Tests locally first, to find potential failures before impacting on the rest of the development process.
3. I aim to Commit every 10 – 15 minutes, and get feedback in < 5 minutes. I wait, patiently for the results of these tests (it's only 5 minutes!) If there are any failures, I (as the committer) will be the best person to understand the problem, find and implement a solution.
4. It is my responsibility to see if my changes are successful. I will monitor progress through all evaluations – fast Unit Tests, and longer-running Acceptance Tests and other evaluations.
5. I aim to fix any failure in 10 minutes. Or if I can't, I will revert the change and work offline to solve the problem, so that I don't impede others' progress.
6. If a team-mate does not stick around to see the results of the tests, or fix a failure, I will revert their change, to keep the route to production open.
7. As a team, we prioritise finding and fixing failures quickly, to keep the software in a releasable state. If it is not clear whose change caused a failure, I get together with the other Committers who could have caused the problem, and we agree who will fix it.
8. The team can encourage adherence to good CI behaviours, by gamifying "Build Sins", such as: wearing a silly hat, putting money in the build sin jar (like a swear jar), and having an instant alert system to notify the whole team about a failure.



9. I recommend adopting TDD – Test Driven Development.
10. I always promote building and using a Continuous Delivery Deployment Pipeline, the best way to organise software development and testing processes into one efficient 'machine'.

If you would like to learn more about Continuous Integration, TDD, and Deployment Pipelines, I have several videos on my Continuous Delivery YouTube channel, https://bit.ly/CDonYT
Or take a look at my on-line courses: https://bit.ly/DFTraining