# Continuous Integration

THE LINUX FOUNDATION

# Why Continuous Integration?

- Once upon a time, most software was written by a relatively small group of developers, often working in the same location and in frequent contact; coordination and division of responsibilities was straightforward
- **Revision control systems** were developed to accommodate more than one contributor working on a project
  - Central **repository** stores the master copy of the project; one or more developers possess the ability to make changes and then check them in
- The Linux kernel was the first really huge distributed development project, and its creator, Linus Torvalds, invented the **git** system for rationalizing distributed development

# Why Continuous Integration? (Cont.)

- A revision control system does not solve the problem of making sure what a diverse group of contributors is doing actually works together; that one set of new code or bug fixes does not conflict with another - this can only be done by **testing**
- Testing requires the following considerations:
  - Can overlapping sets of changes be applied simultaneously, or do they conflict?
  - Does the project compile when all changes are applied?
  - Does it work on all possible targets?
  - What does working mean?
  - Are there non-trivial test suites that can exercise a representative workload enough to give confidence things are fine?
- Continuous integration techniques ensure that testing is so frequent that any problems cannot persist for long; distributed developers stay on the same page.

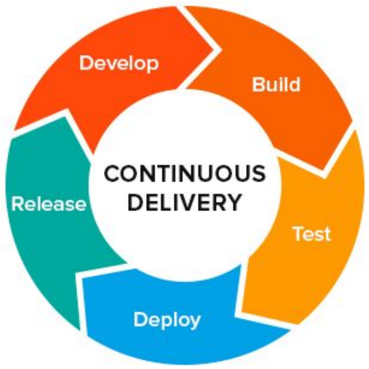
# Continuous Integration, Continuous Delivery, Continuous Deployment

We can distinguish three separate steps/stages:

- **Continuous integration** - changes merged into the main branch ("master") as often as possible; automated builds run on as many variations of software and hardware as possible; conflicts are resolved as soon as they arise
- **Continuous delivery** - the release process is automated and projects are ready to be delivered to consumers of the build; thorough testing is done on all relevant platforms
- **Continuous deployment** - the product is released to customers; again, in an automated fashion

Note: Continuous integration can be considered to include both delivery and deployment.

# Continuous Integration, Continuous Delivery, Continuous Deployment (Cont.)



The time gap between these steps is meant to be as close to zero as possible. In a perfect word, developer changes can reach end user customers the same day or even in minutes.

# Costs and Benefits

| Costs | Benefits |
|---|---|
| Changes have to be merged very often, putting a possible strain on developers | Developers don’t go down the wrong path and compound fixable mistakes, or get in each other’s way |
| The repository must be monitored by a continuous integration server; staff has to be allocated to do this | The build steps are fully automated; all the work has been done up front, instead of each time build testing needs to be done |
| Scripts and other tools have to be run to perform automated tests, report their results and take appropriate actions - it can be a lot of work to prepare this infrastructure | Regressions (bugs which break working product) may be minimized; releases should have fewer bugs |

# Tools

- There are many well-developed continuous integration software tools including:
  - Jenkins (the most widely used)
  - Travis
  - TeamCity
  - GO CD
  - GitLab CI
  - Bamboo
  - Codeship
  - CircleCI
- Some of these products are free in cost, others are not