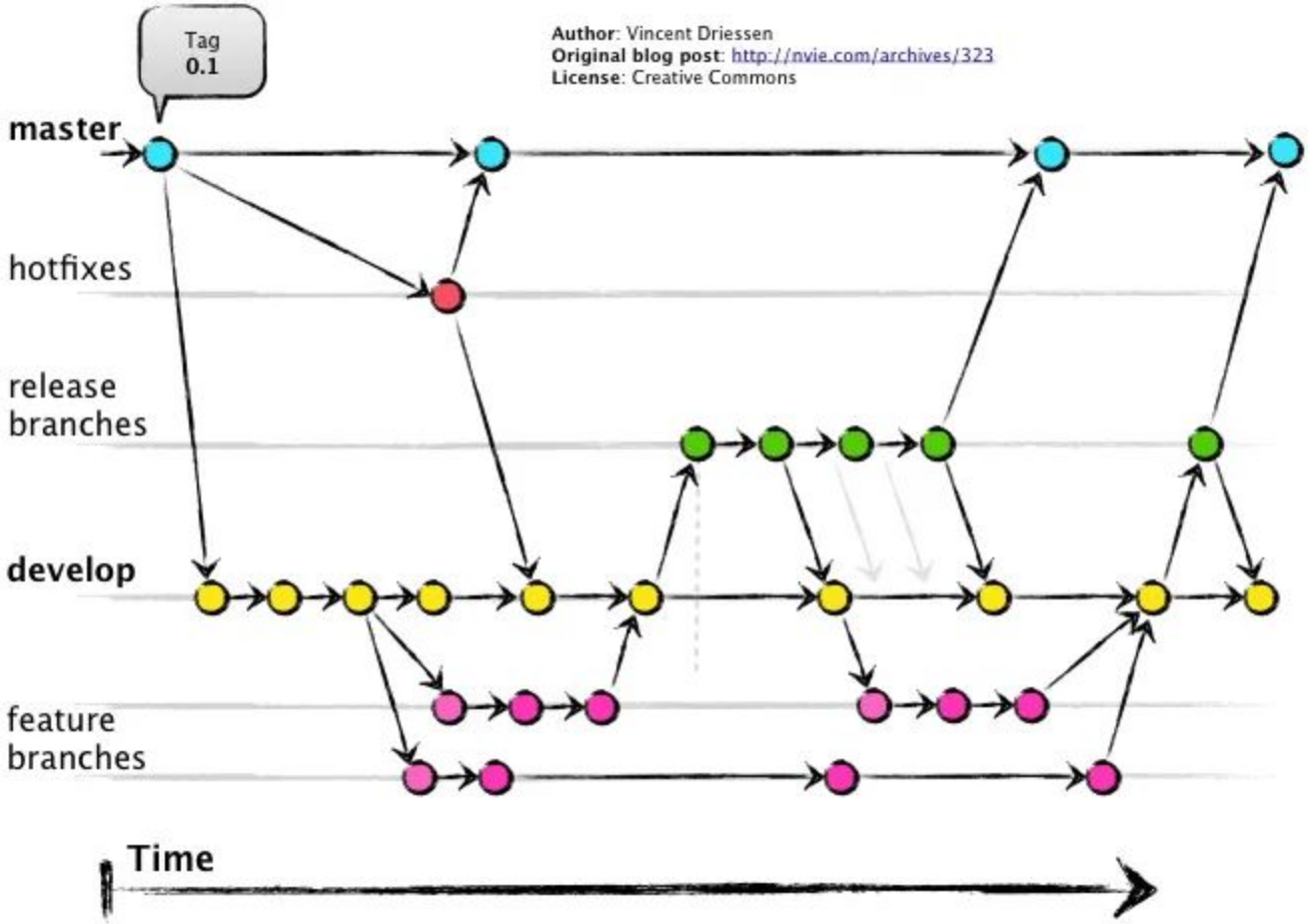
Deployment

Version control

- How to manage changes to code?
- Retain backups of old code
- Develop new features
- Fix bugs


Tag
0.1
Author: Vincent Driessen
Original blog post: http://nvie.com/archives/323
License: Creative Commons
master
hotfixes
release
branches
develop
feature
branches
Time

Version control

- Centralized
  - central server, many clients
  - push changes to server each time
  - multiple editors? Lock files? Merge?
- Distributed
  - can have central server but not needed
  - changes managed using “patches” - email, merge requests, …
- github, gitlab etc.
  - centralized on top of distributed
  - friendly interfaces
  - worth learning command line

Continuous integration

“practice of automating the integration of code changes from multiple contributors into a single software project”

- Atlassian documentation

CI workflow

- Integrate with version control
- Multiple authors contribute to different parts of code
- Central “build server” automatically compiles/builds code

Automation is the key here

Best practices

- Test driven development
  - Write tests before code
- Code review
  - Pull and merge requests - enabled by web interfaces like github/gitlab
  - Review code for correctness, cleanliness, style, …
- Integration pipeline optimization
  - Tests run on each push to server - can be several times a day
  - Fast runs, optimized based on changes etc.

Continuous Delivery / Deployment

- CI/CD - parts of “DevOps” pipeline
- CI = Continuous Integration
- CD could be
  - Continuous Delivery
  - Continuous Deployment

Continuous Delivery

- Once CI (testing) passed, package files for release
- Automated delivery of “release package” on each successful test
- Why?
  - Nightly builds
  - Beta testing
  - Up-to-date code version

Continuous Deployment

- Extend beyond Delivery: Deploy to production
- Passed tests -> deployed to users
  - Users see latest version that has passed tests
  - No installing new versions / updating code or servers
- Benefits
  - Immediate fixes, upgrades
  - Latest features deployed immediately
- Drawbacks
  - Tests may not catch all problems!

Containers

- What?
  - self-contained environment with OS and minimal libraries - just enough to run process
  - Primarily used with Linux kernel namespaces, others like chroot possible
- Why?
  - Full OS impossible to version control - too much software, too many versions
  - Create self-contained images that can be version controlled
  - Sandboxing - image cannot affect other processes on system
- How?
  - Kernel level support needed
  - All communication “inter-container” - networking

Containers

- chroot
  - custom filesystem for part of the code
  - no real process isolation
- FreeBSD jails, Linux VServer, OpenVZ
  - containers in Linux - same kernel, different filesystems
- Control Group namespaces (cgroups) - Linux kernel 2008
  - process isolation through namespaces
- docker
  - mechanisms for managing images - popularized containers
  - problems: bad practices, version control difficult etc.

Orchestration

- App consists of multiple processes, not just one
- Start in some specific order (dependencies)
- Communicate between processes that are isolated
  - Network
- Mechanisms to build and orchestrate, automate
  - docker-compose
  - Kubernetes
- Key to understanding and managing large scale deployments

Summary

- App: idea to deployment
  - Requirements - Tests - Code - Integration - Delivery - Deployment
  - Scaling
- Mechanisms
  - HTML + CSS + JS - Frontend user interface
  - Databases, NoSQL, cloud stores - Backend
  - Authentication, proxying, load balancing - “middleware”
  - Platform-as-a-Service - deployment and change management