

INTEGRACIÓN Y ENTREGA CONTINUAS

Integración Continua



*I've never lost a game,
I just ran out of time.*

Michael Jordan



	Conceptos básicos
Control de versiones	git, cvs, subversion, mercurial, etc.
Repo	<code>uca-gii/construccion</code> alojado en github
Mainline	estado actual del repositorio
Working copy	copia local del repositorio
Check out	clonar el repositorio en local

¡Cuidado! A diferencia de otros SCV antiguos, hacer `checkout` en git es cambiar de rama o restaurar los ficheros de un *working tree*.

Ejemplo de CI a escala

Desarrollo de una nueva característica o *feature*...

Check-out

Hacer **check-out** de una **working copy** en un **repositorio**

```
$ git clone https://github.com/sistemas-sw/construccion
Clonando en 'construccion'...
remote: Enumerating objects: 688, done.
remote: Counting objects: 100% (104/104), done.
remote: Compressing objects: 100% (74/74), done.
remote: Total 688 (delta 45), reused 84 (delta 29), pack-reused 584
Recibiendo objetos: 100% (688/688), 39.75 MiB | 22.49 MiB/s, listo.
Resolviendo deltas: 100% (296/296), listo.
```


Cambiarse al repo con la *working copy* local

```
$ tree -d construccion
construccion/
├── docs
├── marp
├── slides
│   ├── devops
│   │   ├── docker
│   │   │   ├── docs
│   │   │   └── img
│   │   ├── img
│   │   ├── jenkins
│   │   │   ├── entregable
│   │   │   └── img
│   │   ├── scv
│   │   │   └── img
│   │   └── terraform
│   │       └── img
│   ├── implementacion
│   │   └── img
│   └── marp
$ cd construccion
```

Entorno de construcción

Preparar el entorno de construcción:

```
$ cd marp
$ npm install --save @marp-team/marp-core

added 304 packages, and audited 305 packages in 3s

34 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Construcción (*build*)

Construir en local (debería automatizarse):

```
$ cd ..  
$ mkdir ./html/img  
$ cp -R slides/devops/img/ ./html/img/  
$ marp --allow-local-files --config-file ./marp/marp-engine.js \  
  --html slides/devops/cultura.md \  
  -o ./html/cultura.html  
[ INFO ] Converting 1 markdown...  
[ INFO ] slides/devops/cultura.md => html/cultura.html  
$ open ./html/cultura.html
```


Ignorar el *build* (carpeta `html/`) al sincronizar el repo:

```
$ cat .gitignore
...
# Ignorar la carpeta html/
html/*.html
html/img/*.png
html/img/*.gif
**/html
...
```

Desarrollo y pruebas

A partir de ahora:

- Modificar *source code* (ficheros `.md`) para hacer una tarea.
- Hacer tests de que el cambio funciona.
- `git add`, `git commit` y `git push` para subir los cambios al repositorio.

Con git:

- El *index* guarda una instantánea del contenido del *working tree*.
- Hacer *commit* es grabar los cambios en el repositorio (local).
- Hacer *push*: es subir al repo global

Colaboración con otros

Si alguien modifica algo...

```
$ git pull
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 15 (delta 9), reused 15 (delta 9), pack-reused 0
Desempaquetando objetos: 100% (15/15), 3.03 KiB | 344.00 KiB/s, listo.
Desde https://github.com/sistemas-sw/construccion
    23141b5..7cc4304  master    -> origin/master
Actualizando 23141b5..7cc4304
Fast-forward
 slides/devops/cicd.md      | 194 ++++++-----
 slides/devops/cultura.md   | 184 ++++++-----
 2 files changed, 118 insertions(+), 260 deletions(-)
```

Integración

Aún no hemos acabado... Hay que hacer un build (manual o automático) en un servidor de integración común (v.g. Jenkins, Github Actions).

- Si hay un conflicto entre dos desarrolladores, se suele detectar cuando el segundo hace un build sobre su copia de trabajo. Hay que arreglarlo lo antes posible.
- El repo debe quedar en todo momento con un software estable, funcional y con pocos errores.
- No hay que alejarse mucho de esa base estable pues llevaría mucho tiempo integrarse con ella.

Martin Fowler: [Building a feature with CI](#)



Beneficios de CI

Abordar problemas del desarrollo:

- Software complejo
- Cambios inesperados e incompatibles
- Desarrollo en equipo

Ventajas:

- Retroalimentación rápida
- Lotes pequeños
- Calidad y productividad

Prácticas de CI

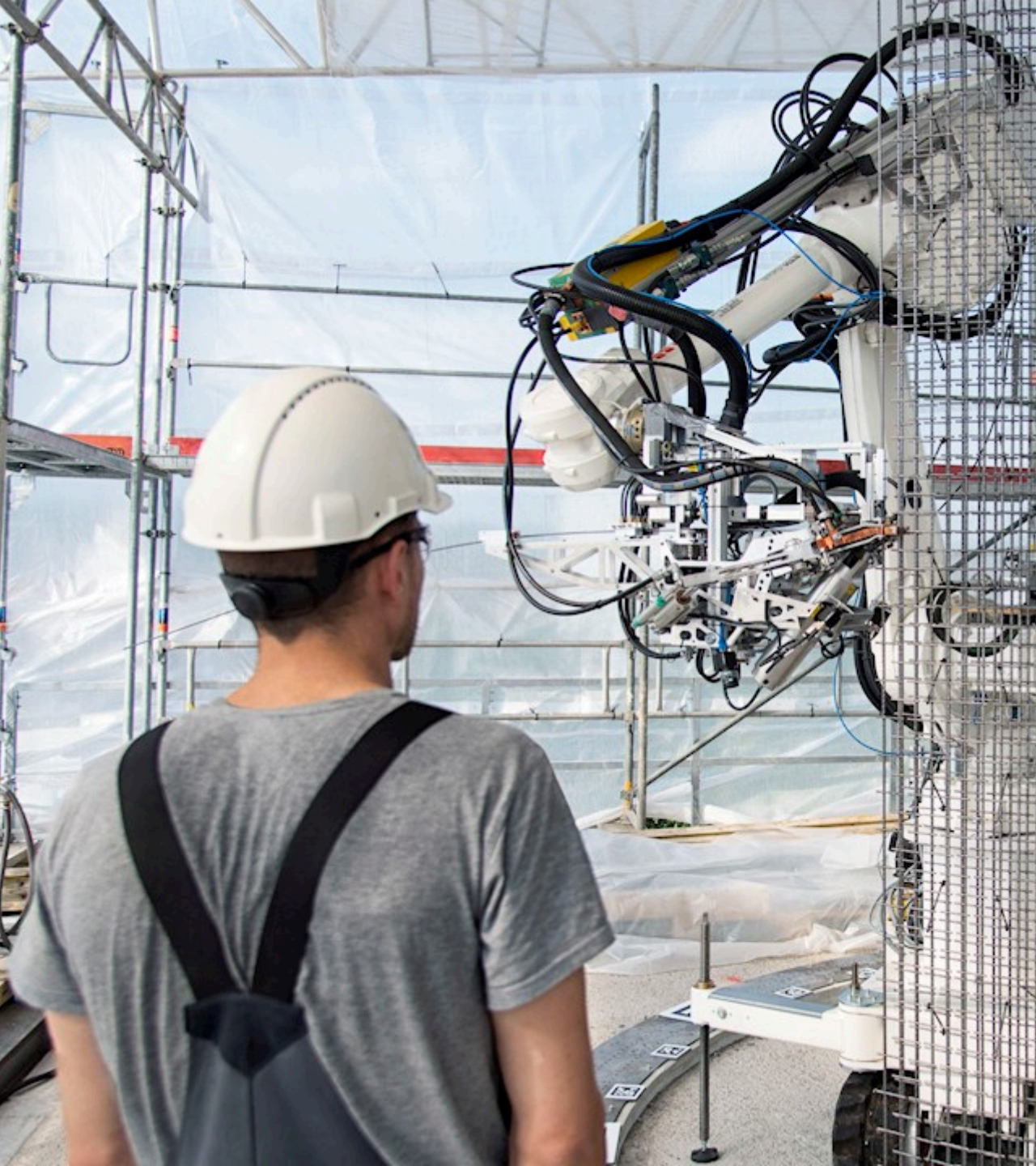
- Un solo repositorio de código fuente
- Automatizar la construcción (build)
- Hacer el build self-testing
- Todos deben hacer commit al trunk todos los días
- Cada commit a la *mainline* debe originar un build en un servidor de integración
- Arreglar inmediatamente los builds fallidos
- Mantener rápidos los build
- Etc.

Martin Fowler: [Practices of CI](#)

Un solo repositorio

- SCM, CVS, configuration management,...
 - Todo en el **monorepo**...
 - ...menos los productos del build
- Instalar SO, [IDE, SGBD] y... checkout!
- Minimizar número de ramas





Automatizar los *build*

- Build tools: make, GNU Autotools, Apache ant, mvn, gradle, dotnet msbuild, Ruby rake, etc.
- Dependencias: Apache ivy, maven, npm, yarn, pip, conda, NuGet, cargo, etc.

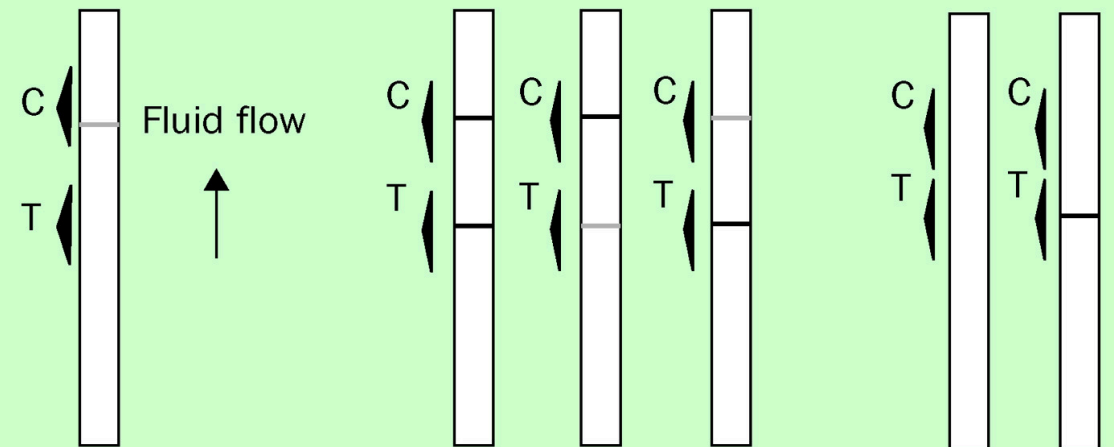
No depender de los IDEs para hacer builds

Hacer el build self-testing

- ¿Hacer TDD o XP?
- Suite de tests automatizados
- XUnit, UI tests (Selenium), APIs (Appium), Mocking (mockito), etc.
- SAST (SonarQube, ESLint, etc.)



Self-test interpretation



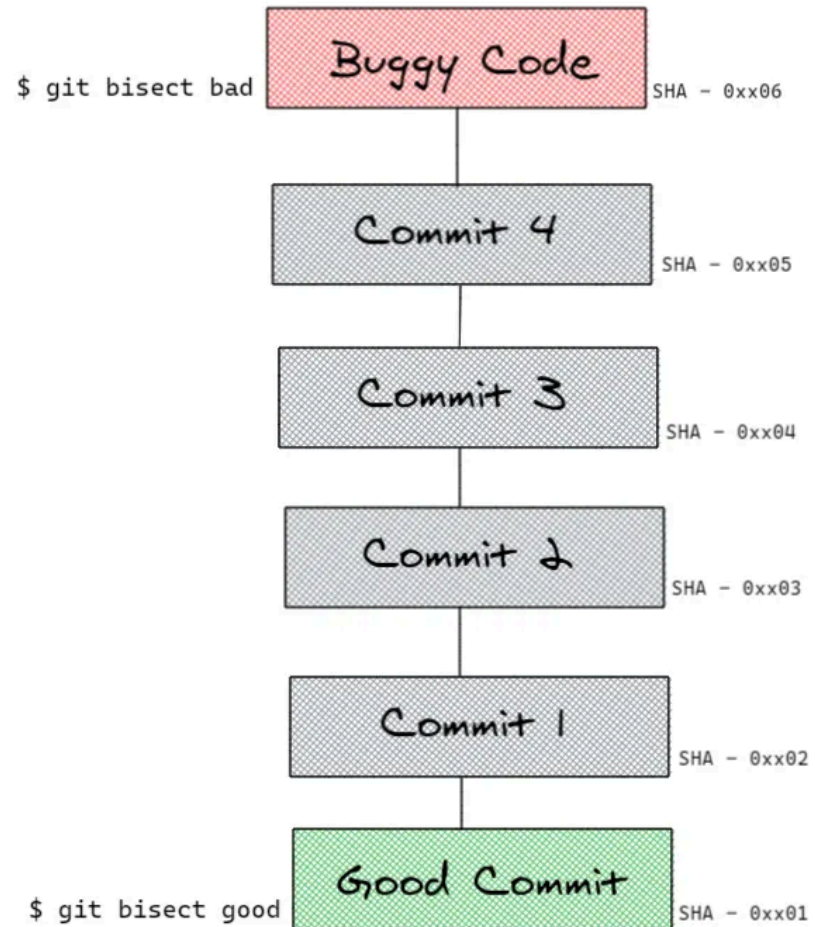


Todos deben hacer commit al trunk todos los días

- Encontrar problemas pronto
- Frecuentes merge de ramas en el trunk
- Diff debugging tras detectar fallos al ejecutar el build:

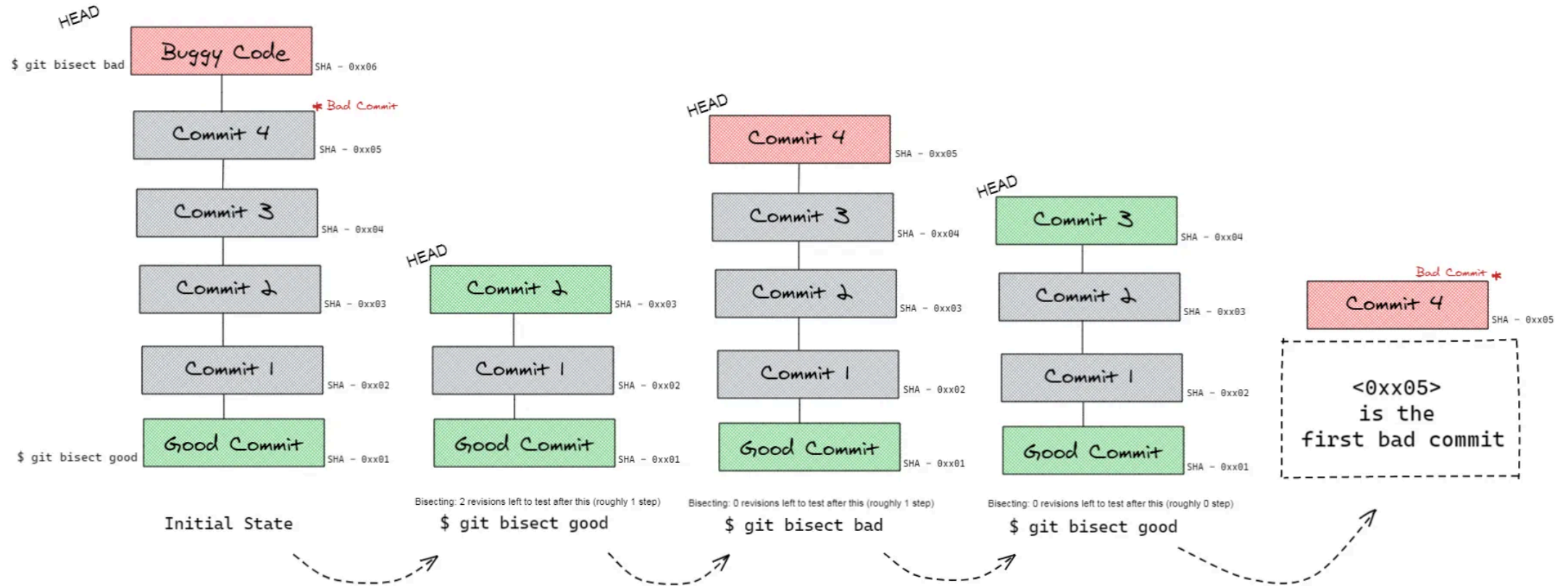
`git bisect`

Búsqueda binaria de bugs en *commits* (inicio)



```
$ git bisect start
$ git bisect bad
$ git bisect good <0xx01>
```


Búsqueda binaria de bugs en *commits* (repetir pasos)



`$ git bisect reset`

1 commit de *mainline* $\Rightarrow$ 1 build en servidor de integración

- Update y build local... ¡no siempre se hace!
- Pues en mi máquina me funciona...
- Build en máquina compartida (manual vs. servidor de integración)
- Servidores de CI: Jenkins, Gitlab CI/CD, Teamcity, Bamboo, GitHub Actions, Azure DevOps services, CircleCI, Semaphore,





Arreglar inmediatamente los builds fallidos

- Prioridad 1
- Un par de personas basta
- Técnica rápida: revertir el commit más reciente que ha roto el build y debug en local
- Técnica para evitar romper la mainline: crear *working copy* desde *head* y hacer *commits* en una rama **pending-head** diferente

Mantener rápidos los build

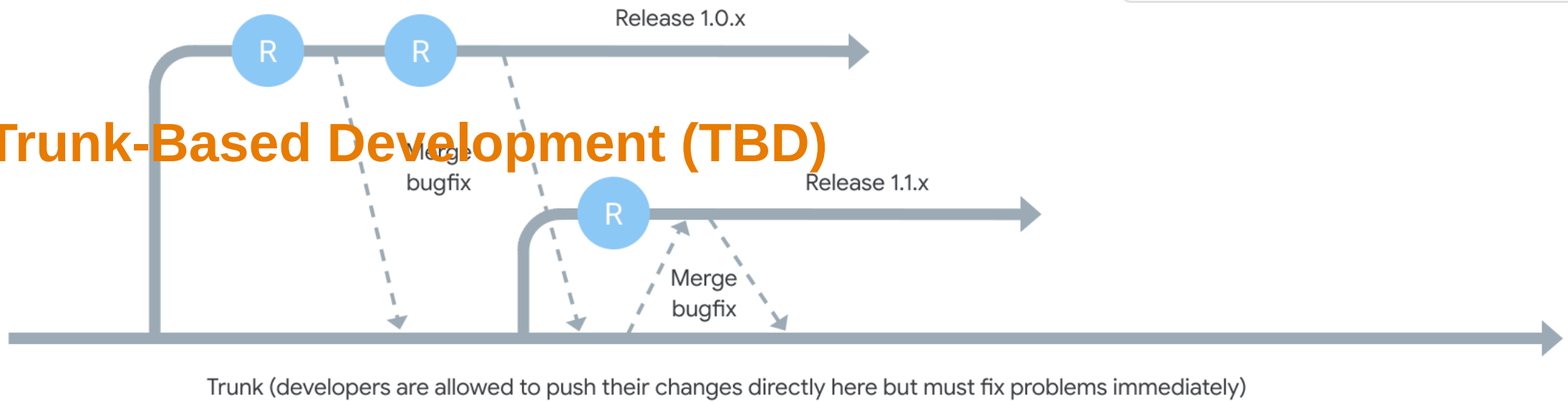
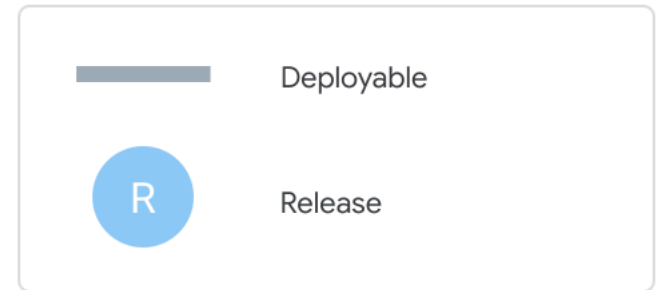
- ¿1 hora es mucho? ¿10 minutos?
- Testing: cuello de botella
- *Build pipeline* o *Staged build*
- Ejemplo: 2-stage pipeline
 - 1ª etapa rápida (compilación y pruebas unitarias sin la BD) → 1er. *commit build*
 - 2ª etapa lenta (pruebas de integración con la BD real) → build secundario → si falla, añadir tests al commit build

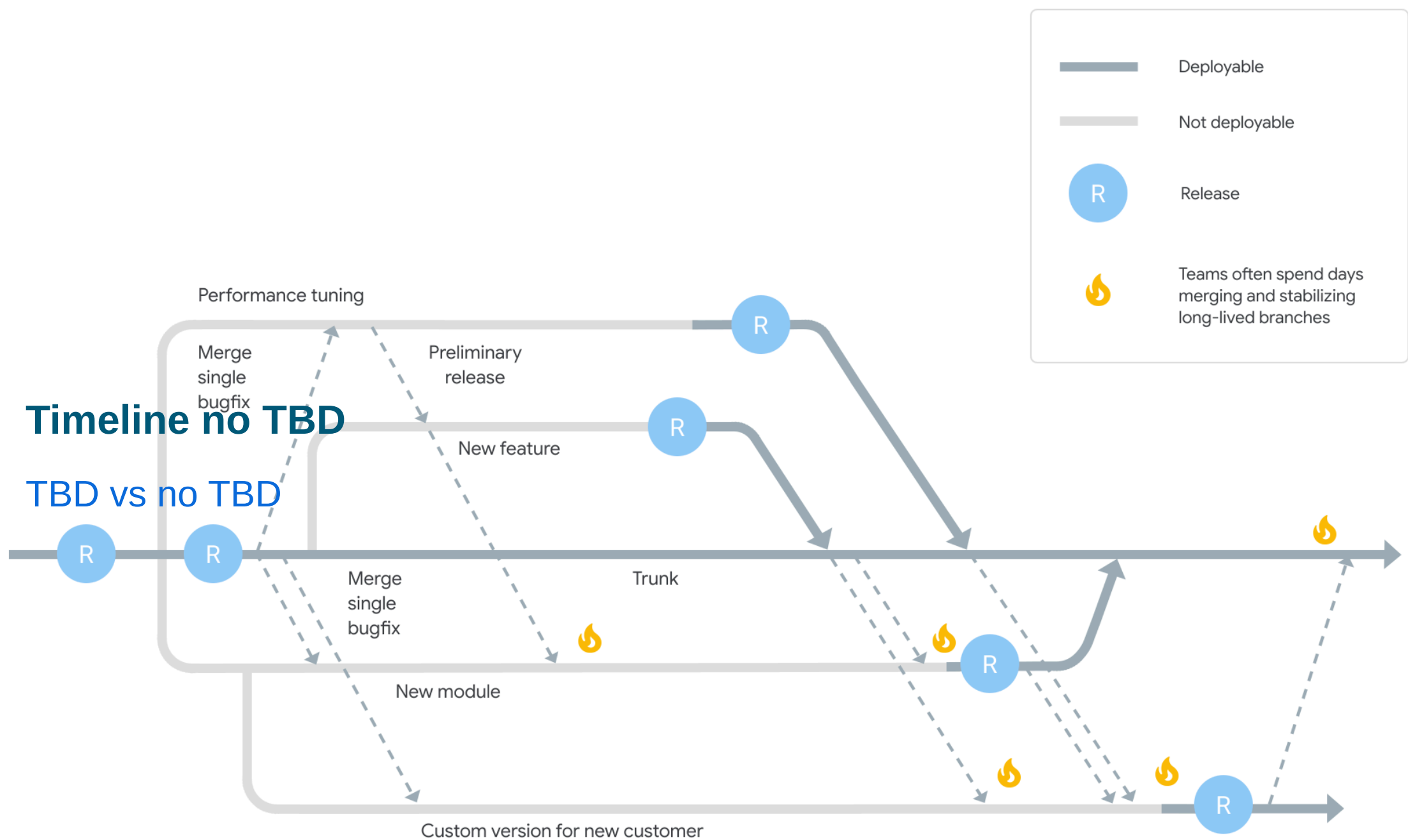


Otras prácticas de CI...

- Testear en un clon del entorno de producción
- Hacer que sea fácil para cualquiera obtener el ejecutable más reciente
- Todos pueden ver lo que está pasando
- Automatizar el despliegue

Trunk-Based Development (TBD)





Controversia de CI

- Prácticas de CI son controvertidas
 - Dividir *features* grandes en pasos pequeños
 - Lleva más tiempo completar las *features* grandes
- Si los cambios son pequeños
 - Desarrollo + Entrega más rápidos y estables
 - Las ramas son de corta duración
 - Los desarrolladores reciben comentarios periódicos sobre el impacto de su trabajo en el sistema en conjunto
 - Más fácil y rápido detectar, clasificar y solucionar problemas

CI es el paso previo a la CD



Entrega Continua



*Lo entregó todo al fuego,
que no hace distinción*

Plutarco

Entrega Continua

Se hace CD cuando:

- Software desplegable en cualquier momento
- Equipo prioriza mantener el software desplegable
- Tras un cambio, cualquiera puede saber rápidamente si el sistema está listo para producción
- Se puede desplegar con un click cualquier versión del software en cualquier entorno



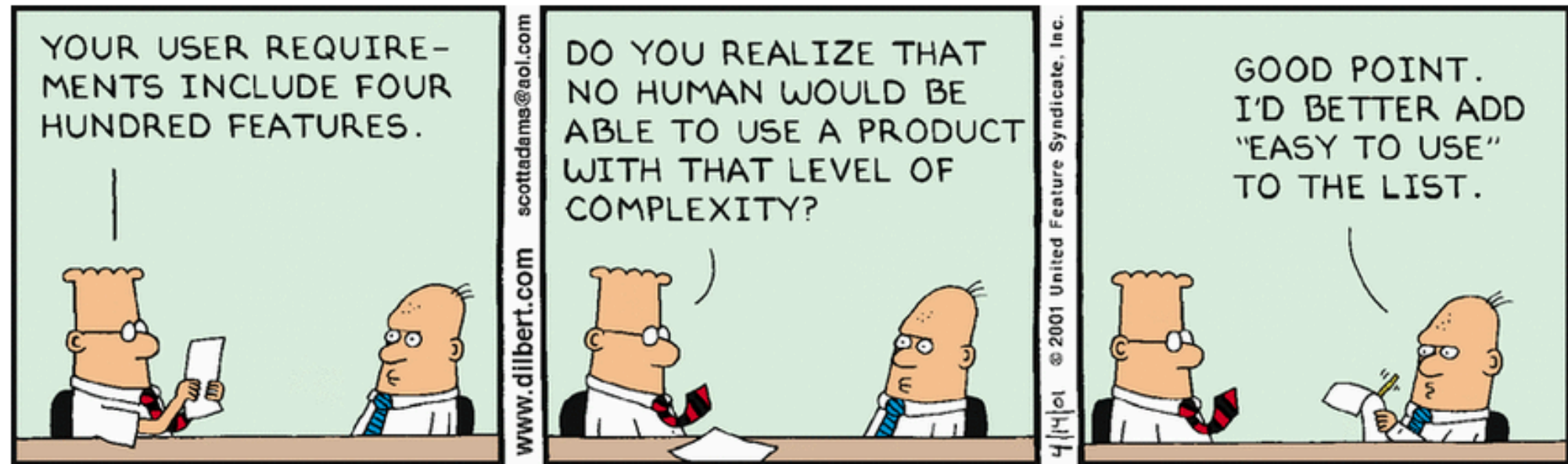
Beneficios de CD

- Despliegues con riesgo reducido
- Progreso creíble: ¿quién garantiza el *done*? ¿que esté en producción? ¿que lo digan los desarrolladores?
- Feedback de los usuarios: reduce el riesgo de construir algo inútil

Cuanto antes te des cuenta...



Saturday April 14, 2001



Cómo implementar CD

- Automatizar el build, las pruebas y el despliegue
- Trunk-based development
 - nº ramas activas < 3
 - ramas y forks con vida corta (< 1 día)
 - pocos locks del código (merge conflicts, freezes, etc.)
- Shift-left de la seguridad
- **Arquitectura poco acoplada**
- Dejar a cada equipo elegir sus herramientas
- Control de versiones de configuraciones y scripts de despliegue
- **Gestión de cambios en la base de datos:** *fixtures*

Errores comunes al implementar CD

- Creer que CD implica hacer despliegues frecuentes
- No hacer cambios en las capacidades técnicas necesarias para hacer CD
- Centrarse solo en herramientas y patrones (v.g. deployment pipeline)
- No hacer CD porque no se puede hacer CDEP

Transformación

