Parallel frontend deployment

for continuous integration and deployment

by Matthias Hryniszak

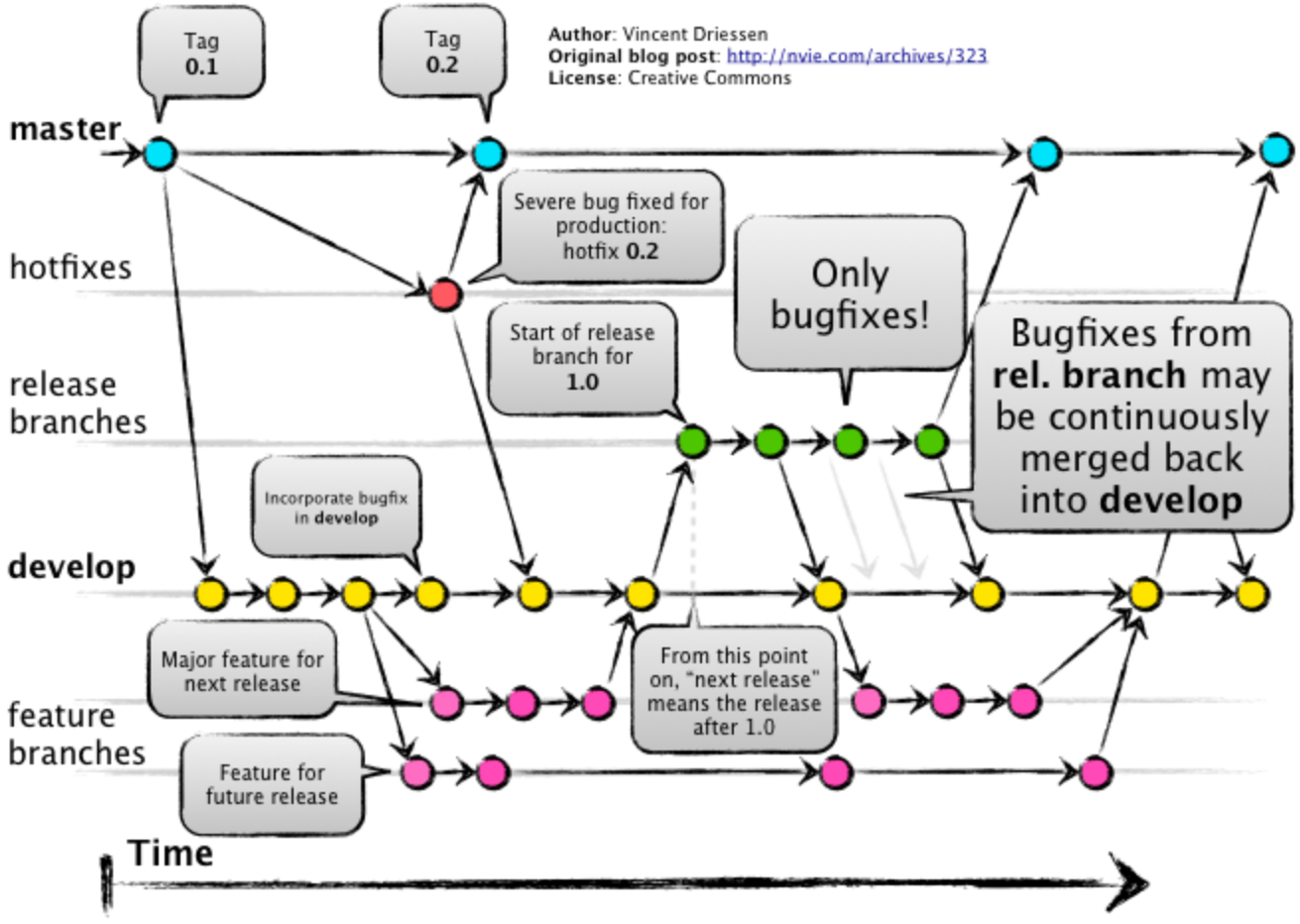
Continuous deployment

Continuous deployment parts

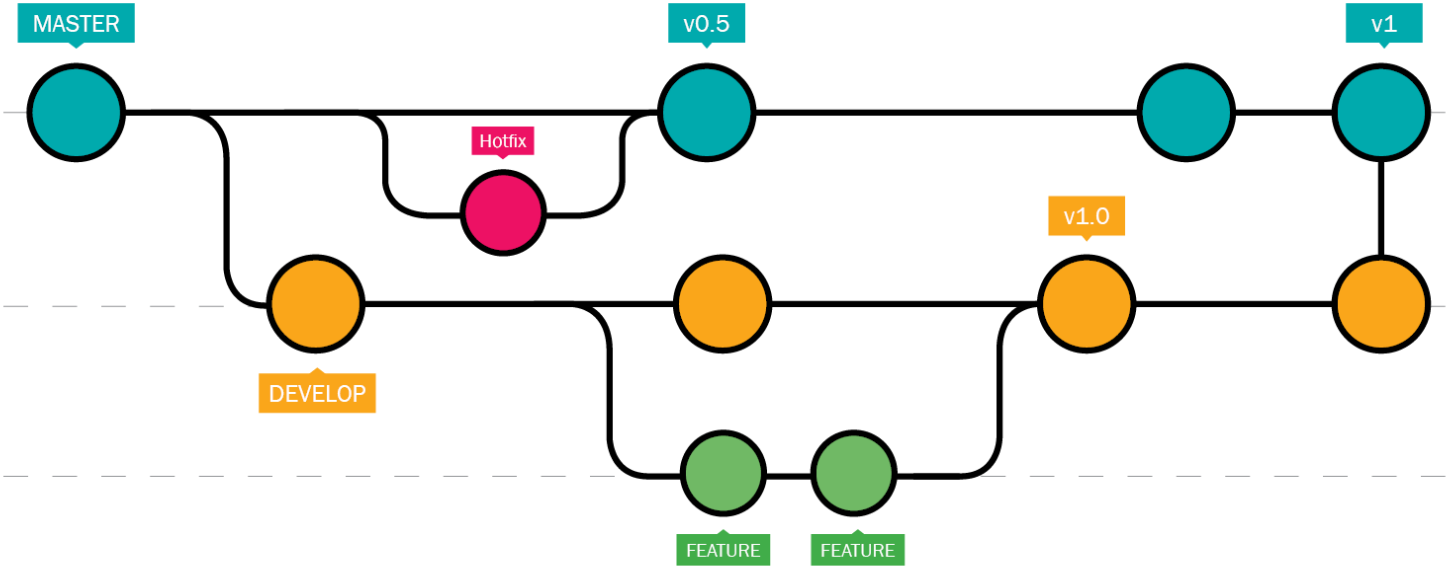
- Version control
- CI/CD automation
- Runtime environment

Version control

Git workflow

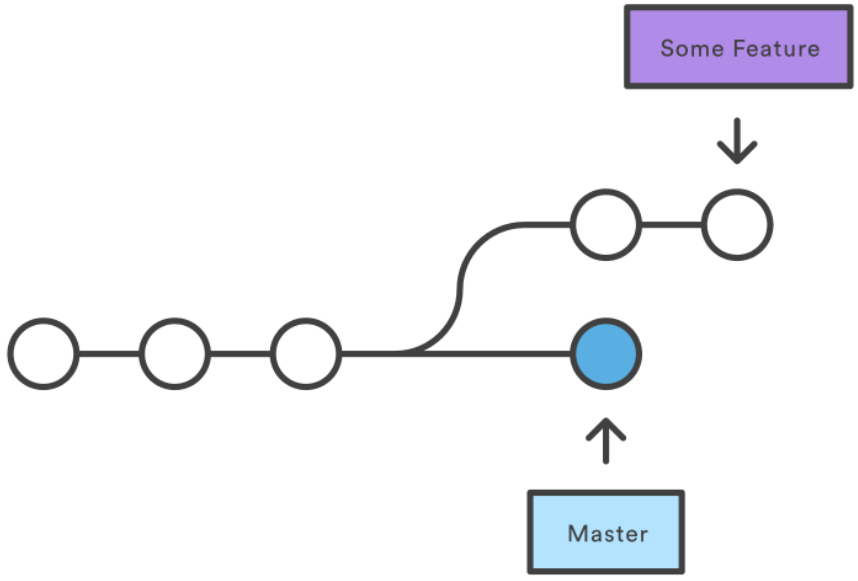


Git workflow - explained



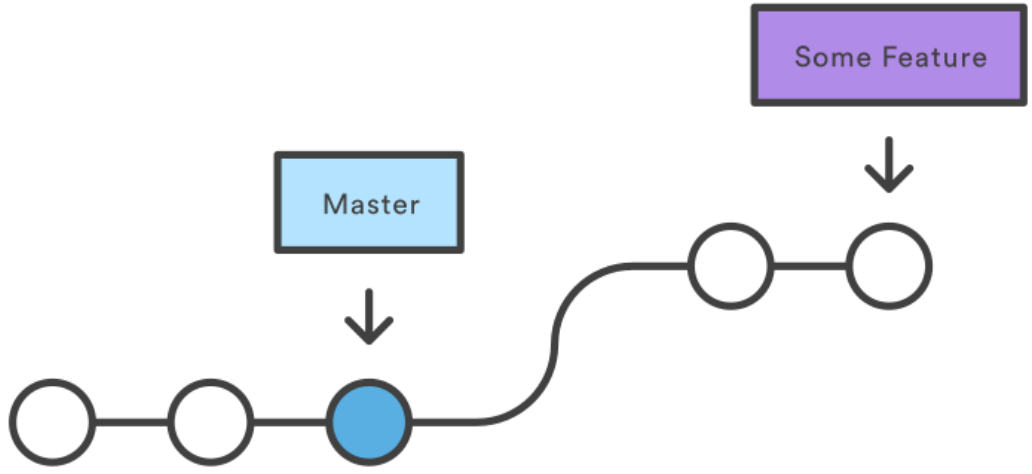
Fixing git workflow

Git workflow - fixed - step 1



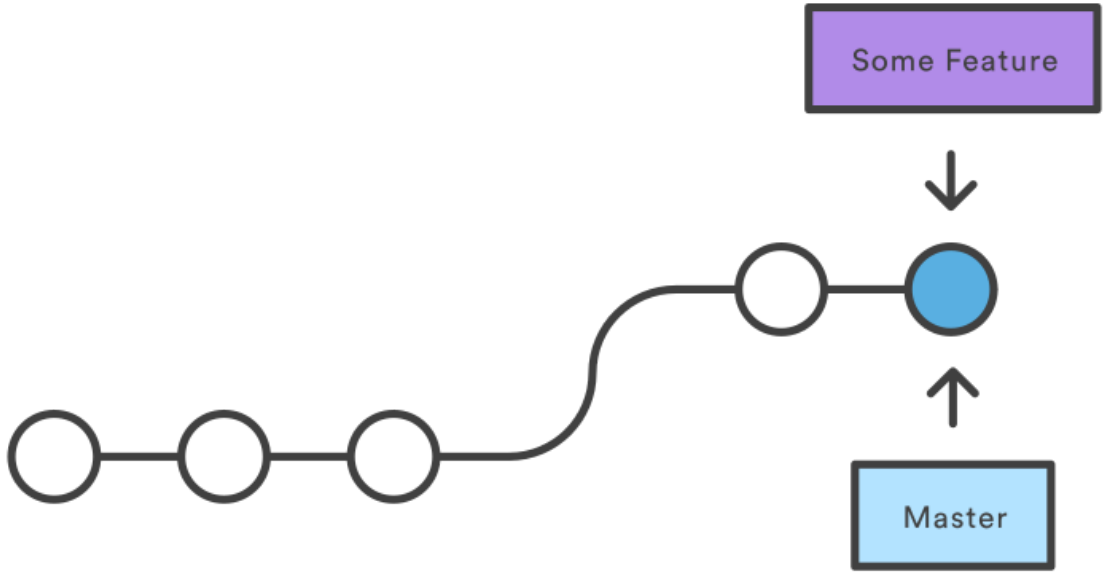

```
$ git checkout -b feature-branch
...
$ git commit
$ git pull --rebase
$ git push
```

Git workflow - fixed - step 2




```
$ git fetch origin
$ git rebase origin/master
$ git push --force
```

Git workflow - fixed - step 3




```
$ git checkout master
$ git pull
$ git merge --ff-only feature-branch
```

CI/CD automation

CI/CD automation

- Jenkins (Pipeline Multibranch Plugin)
- Team City (Working with multiple branches)
- TFS (Build multiple branches)
- Bamboo (Using plan branches)

Runtime environment

Runtime environment - backend

- Multiple processes (HTTP server, queues, database,...)
- Runs in isolation from the client (on the server)
- Runs on managed hardware (or in the cloud)
- Deployment requires restart of server process
- High availability is managed through clever load balancing
- Managing backend version requires clever load balancing
- **Slow change ratio**

Runtime environment - frontend

- One process (the browser)
- Runs in isolation from the server
- Runs on client hardware
- Deployment is potentially done on every page refresh
- **Very rapid change ratio**

Development flow

The development flow

- **Backend first** - fully deployed and tested
- **Then frontend** - building on top of backend services

Parallelizing frontend deployments

Baibulo*

- Node.js (https://www.npmjs.com/package/baibulo)
- Java/JavaEE (com.aplaline.baibulo:baibulo:1.0.6)
- .NET (https://www.nuget.org/packages/baibulo-net)

* "baibulo" means "version" in Chewa

Baibulo - Node.js

```
const app = require('express')()
const cookieParser = require('cookie-parser')
const baibulo = require('baibulo')

app.use(cookieParser())

app.use(baibulo({ root: '/var/lib/my-project' }))

app.listen(3000, () => {
  console.log("Listening for requests on ports 3000\n");
})
```

Usage

Accessing different versions

- Specific branch:
  http://server/path?version=branch-name
- Release version:
  http://server/path
  http://server/path?version=release

Version discovery scheme - GET

- `version` query string param
- `Version` HTTP header
- `version` query string param in `Referer` header
- `__version` cookie
- defaults to `release`

Version discovery scheme - GET

- Once the first request is made the `__version` cookie is set
- Following requests inherit version from the first request

Version discovery scheme - PUT

- `version` query string param
- `Version` HTTP header

Deployment of a file in a version

Using header:

```
$ curl -v -X PUT \
  --data-binary "@image.png" \
  -H "Version: TST-123" \
  http://server/assets/image.png
```

Using query string param

```
$ curl -v -X PUT \
  --data-binary "@image.png" \
  http://server/assets/image.png?version=TST-123
```

Using `baibulo-deploy` utility

```
$ baibulo deploy \
  --dir dist \
  --url http://server/assets \
  --version=TST-123
```

https://www.npmjs.com/package/baibulo-deploy

Security

Upload-only server

```
const app = require('express')()
const baibulo = require('baibulo')

app.use(baibulo({
  root: '/var/lib/my-project',
  download: false,
  upload: true,
}))

app.listen(3001, () => {
  console.log("Listening for uploads on ports 3001\n");
})
```

Download-only server

```
const app = require('express')()
const cookieParser = require('cookie-parser')
const baibulo = require('baibulo')

app.use(cookieParser())

app.use(baibulo({
  root: '/var/lib/my-project',
  download: true,
  upload: false,
}))

app.listen(3000, () => {
  console.log("Listening for requests on ports 3000\n");
})
```

Storage

Storage - how versions are stored on disk

```
/storage-root/folder/filename/version
```

```
/var/lib/my-project/index.html/release
/var/lib/my-project/index.html/TST-123

/var/lib/my-project/styles/styles.css/release
/var/lib/my-project/styles/styles.css/TST-123
```

Implications

Implications

- DevOps can easily implement continuous deployment
- Only one server is needed to host **all** versions
- Product owners can decide if the feature is going as planned
- Testers can **always** test what is **really** going into production
- Developers can easily integrate with upstream changes
- Single point fast-forward final merges without merge conflicts
- **Clear** and **linear** change history on master
- Ability to use `git bisect` to find troublesome commits

Using remote repo browsers testers can actually merge!

Incremental implementation

Incremental implementation

1. Implement versioning on integration servers
2. (optional) Implement versioning on pre-production servers
3. (optional) Implement versioning on production servers

The Future

Planned improvements

- Multiple storage options (S3, Redis, relational databases)
- ETag handling
- Removal of old versions
- Python implementation
- Ruby implementation

Contributions are welcomed!

Questions?

May the force be with you!

Blog:

https://padcom13.blogspot.com

LinkedIn:

https://linkedin.com/in/padcom

This presentation:

https://bit.ly/2Q5hE7z