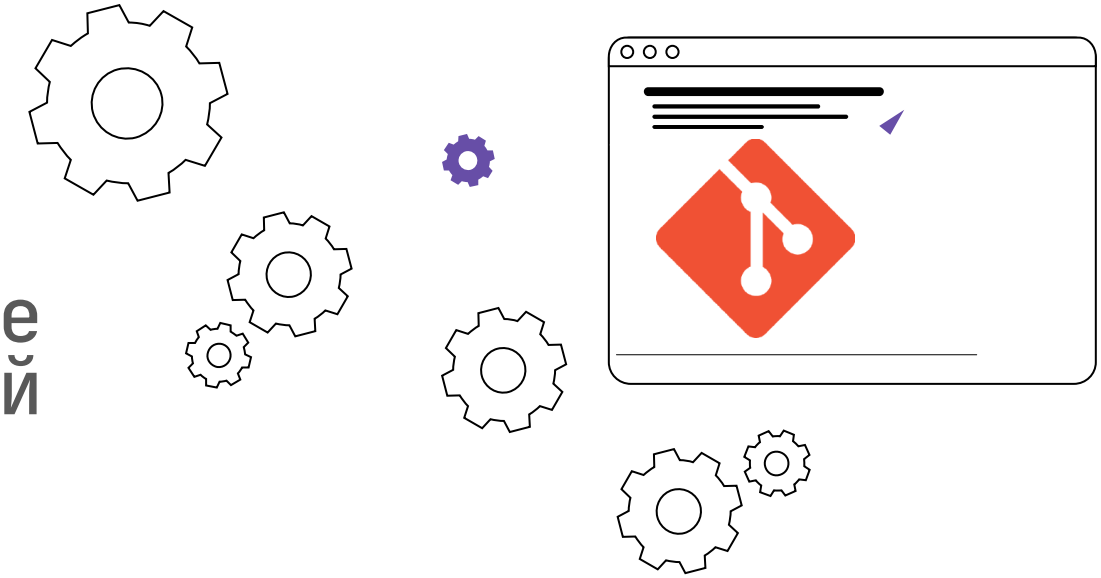
От коммита до релиза:

Версионирование Java-приложений



$ whoami?

JPoint

Joker<?>

Code Fest

sysconf

Андрей Кулешов

intel

Deutsche Bank

HUAWEI

pt

Я

Дисклеймер!

Чат JPoint и Joker — конференций по Java

1 848 members, 880 online

Dima 💎

JPoint и Joker — канал конференций по Java

#видеозаписи Пока мир обсуждает OpenAI, разработчи...

Три всадника апокалипсиса:

- доклад про this в Java
- доклад про OSIV в Spring
- доклад про springdoc

😁 5 👏 1

17:42

Чат JPoint и Joker — конференций по Java

1 848 members, 880 online

Dima 💎

JPoint и Joker — канал конференций по Java

#видеозаписи Пока мир обсуждает OpenAI, разработчи...

Три всадника апокалипсиса:

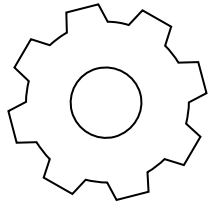
- доклад про this в Java
- доклад про OSIV в Spring
- доклад про springdoc

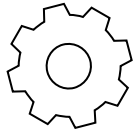
😁 5 👏 1

17:42

+ доклад про git и версионирование

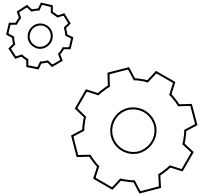
Про что поговорим?

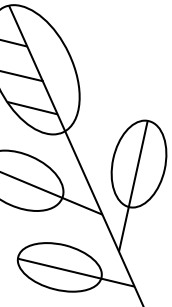




От **коммита**
до **релиза**:

Версионирование
Java-приложений





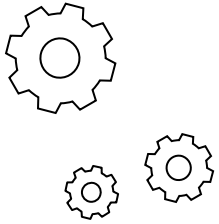
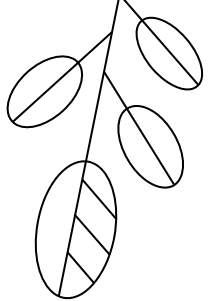
Про что поговорим?

https://github.com/**orchestr7**/vercraft

Про что поговорим?

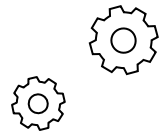
СТРАТЕГИИ

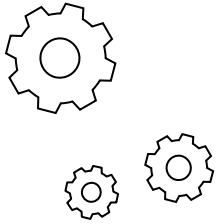
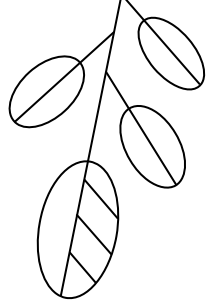

Sylvos Windrunner
Level 8 Blood Mage
Damage:
42 - 48
Armor:
5
Strength:
30
Agility:
22
Intelligence:
40
Status:
733 / 850
600 / 600
Inventory

Про что поговорим?

СТРАТЕГИИ релизов Java-приложений

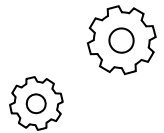



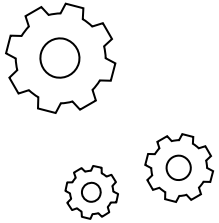
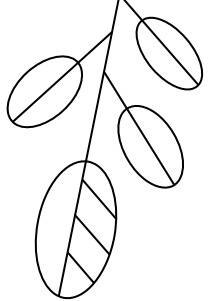



Про что поговорим?

СТРАТЕГИИ релизов Java-приложений

Версионирование и утилиты для этого

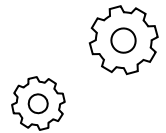



Про что поговорим?

СТРАТЕГИИ релизов Java-приложений

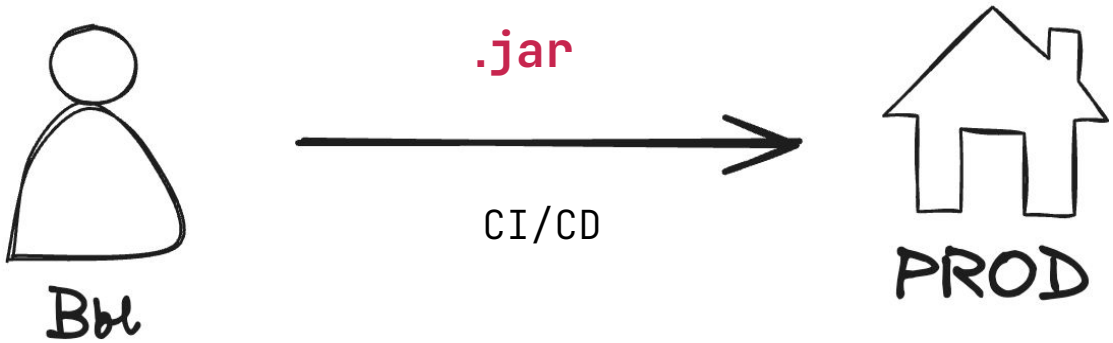
Версионирование и утилиты для этого

Немного про автоматизации: git и Java

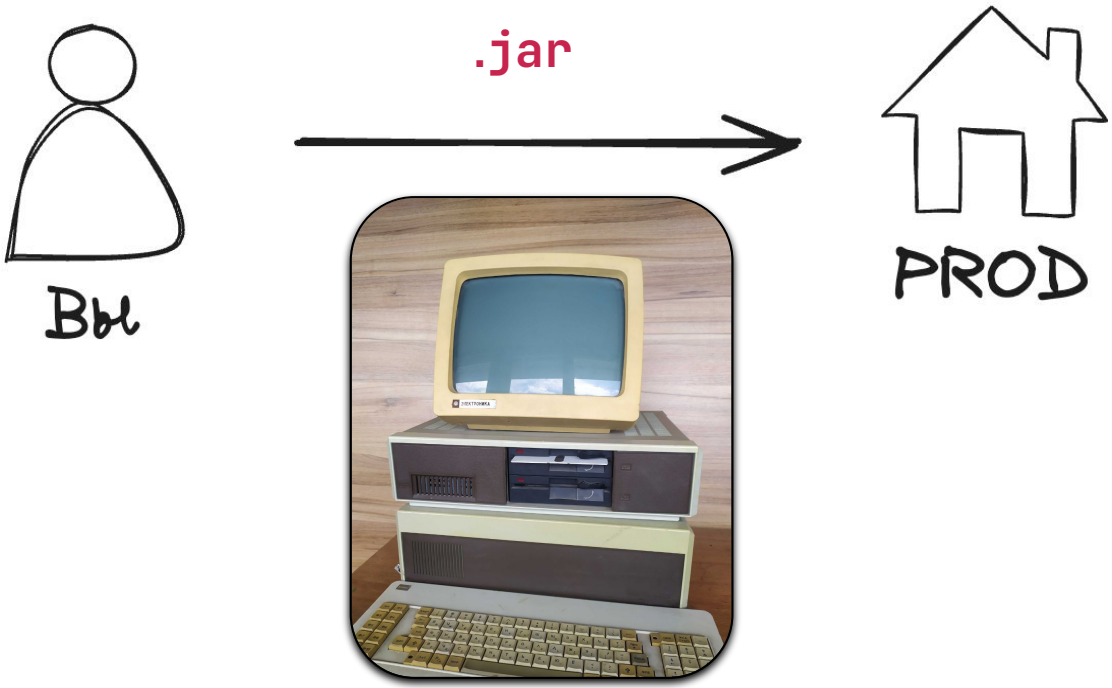



Проблема

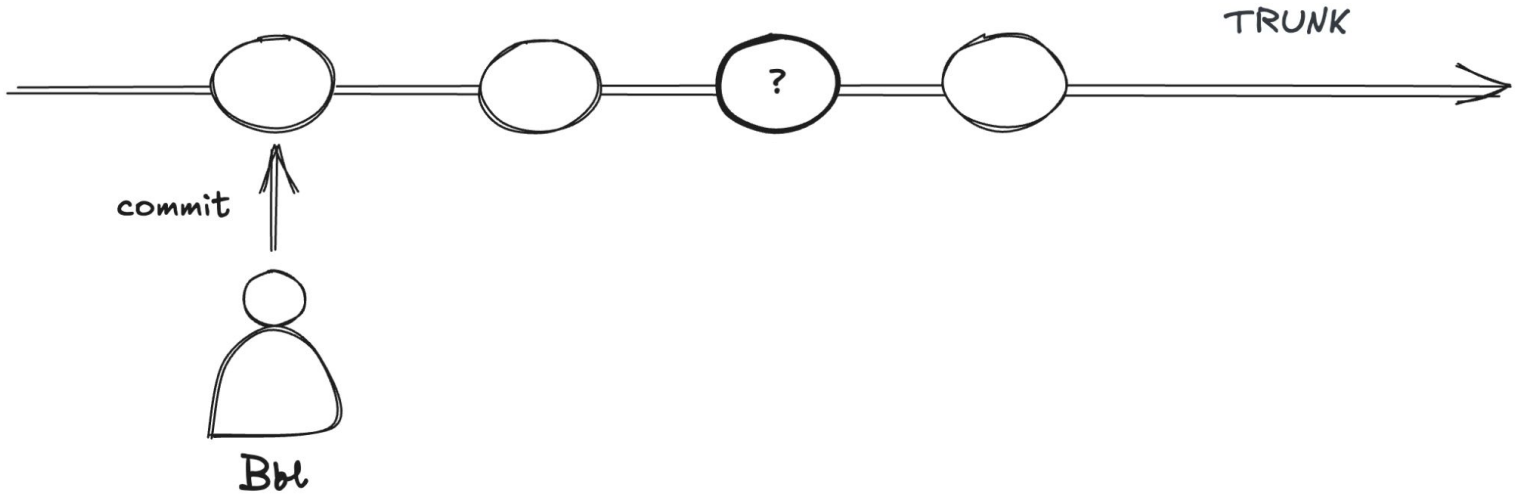
Проблема



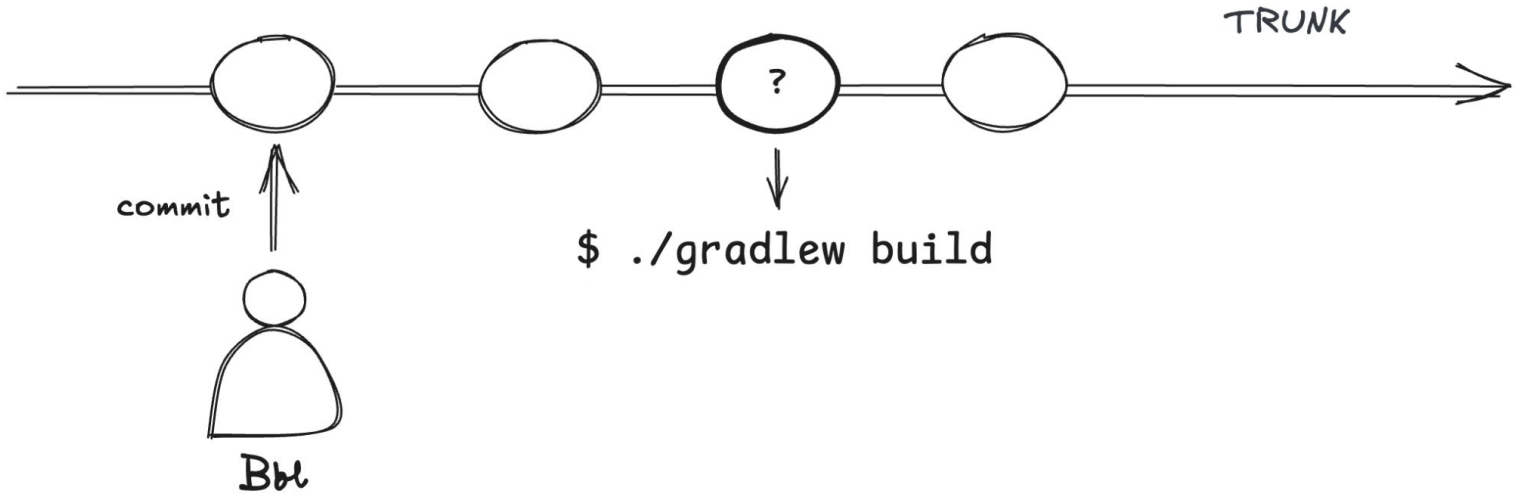
Проблема



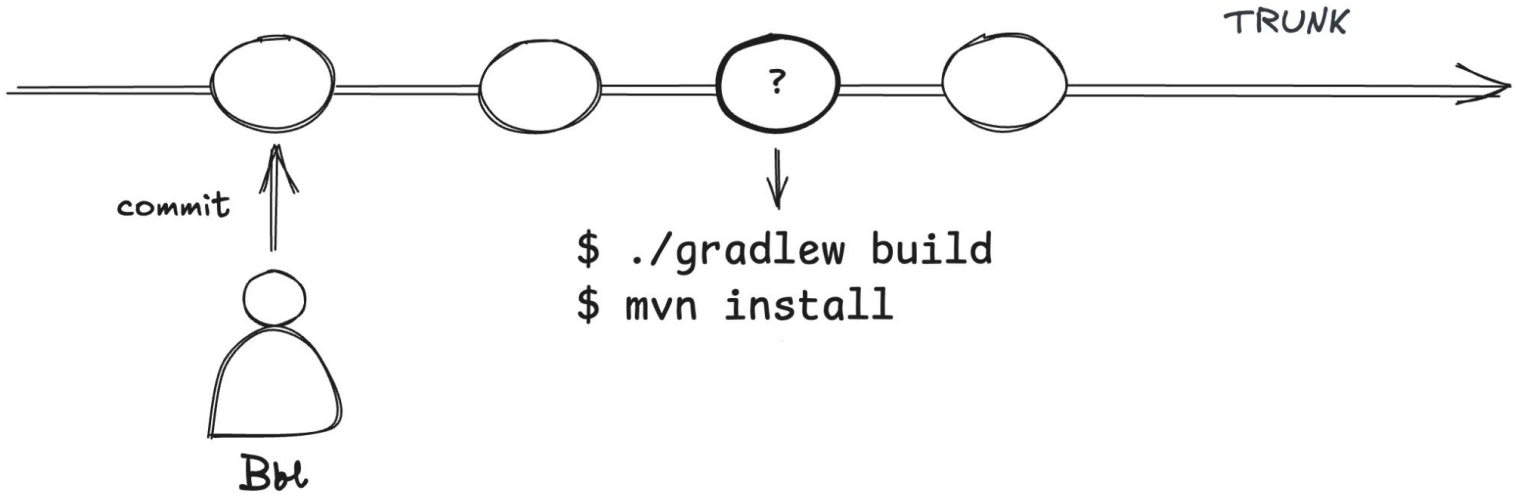
Проблема



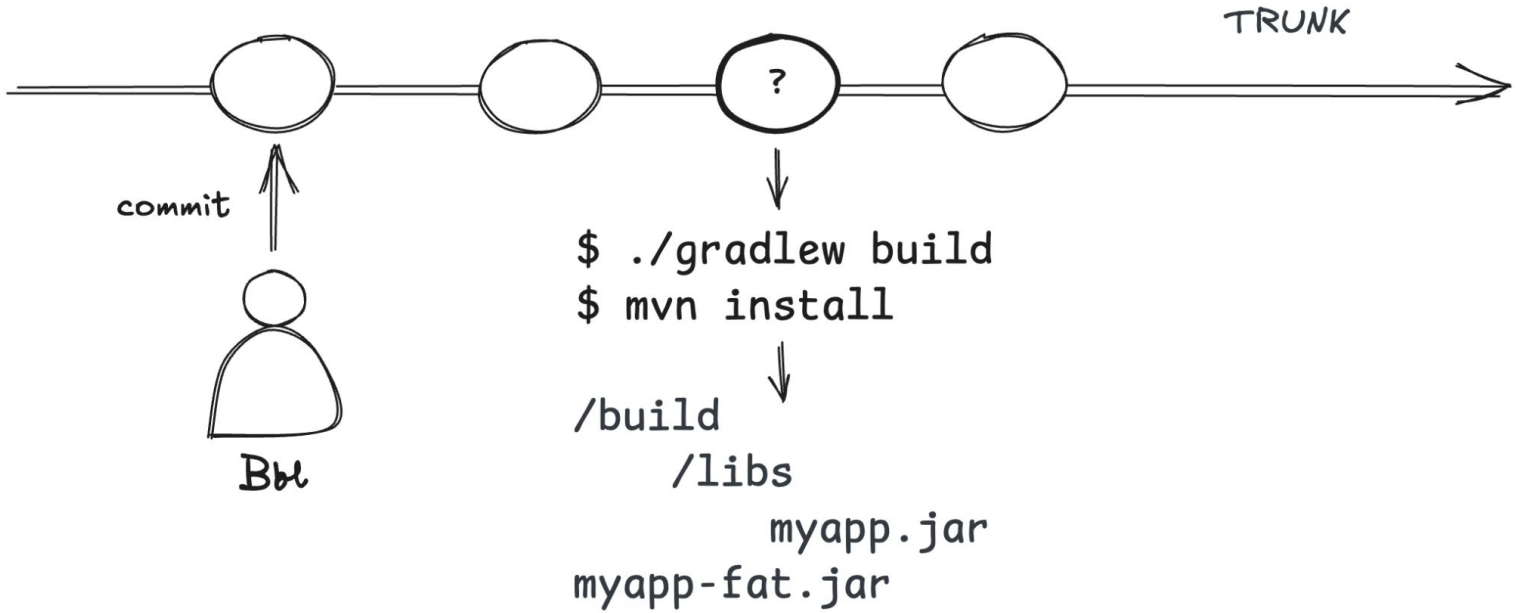
Проблема



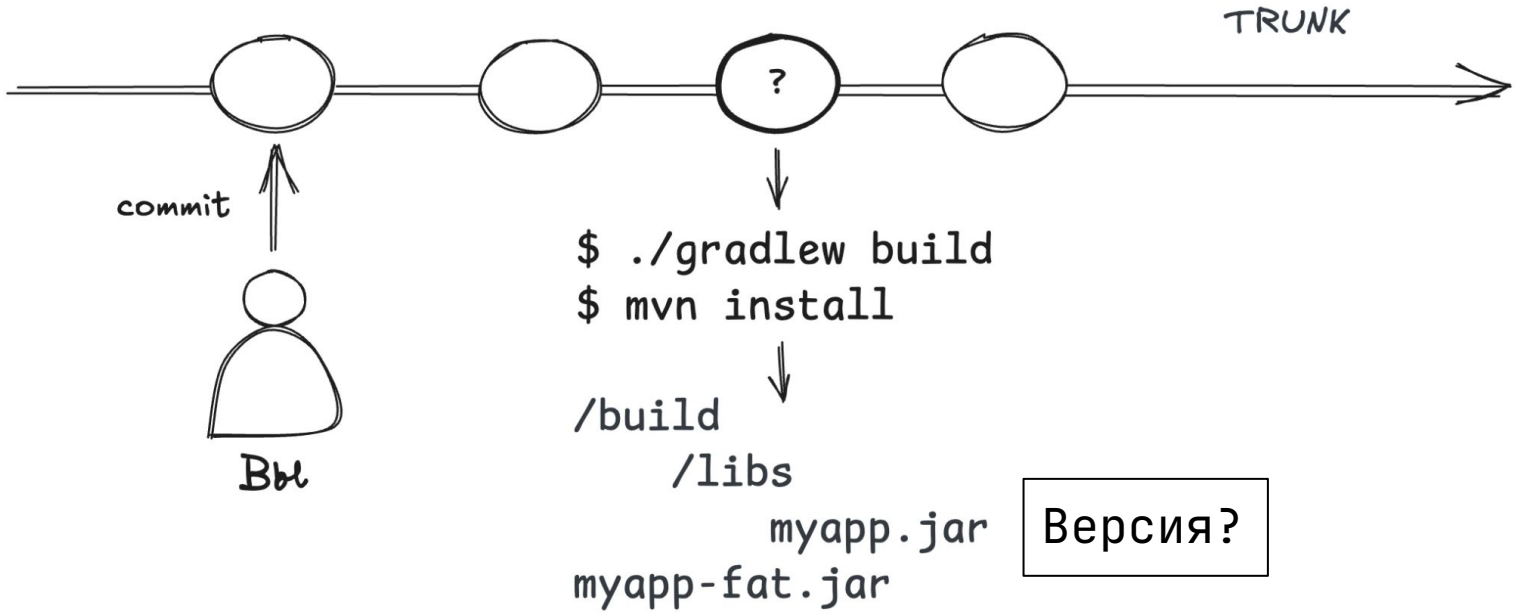
Проблема



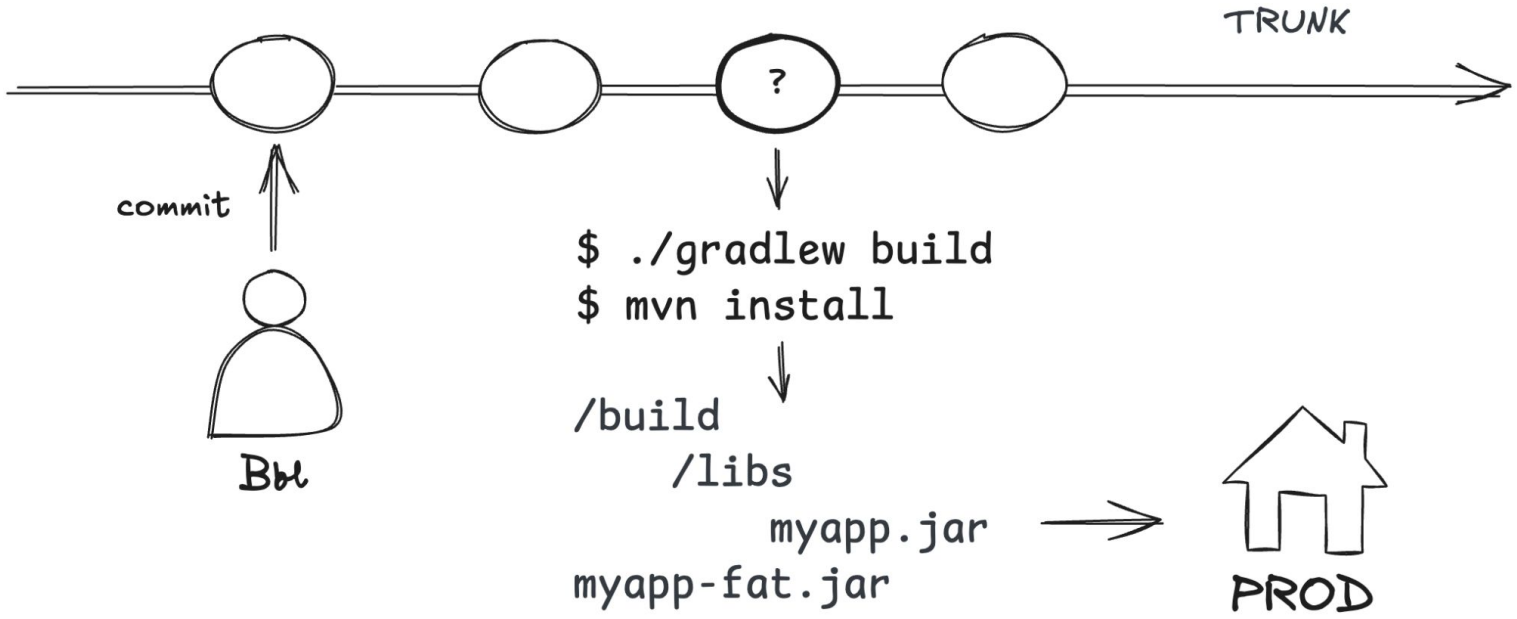
Проблема


TRUNK
?
commit
Вы
$ ./gradlew build
$ mvn install
/build
/libs
myapp.jar
myapp-fat.jar

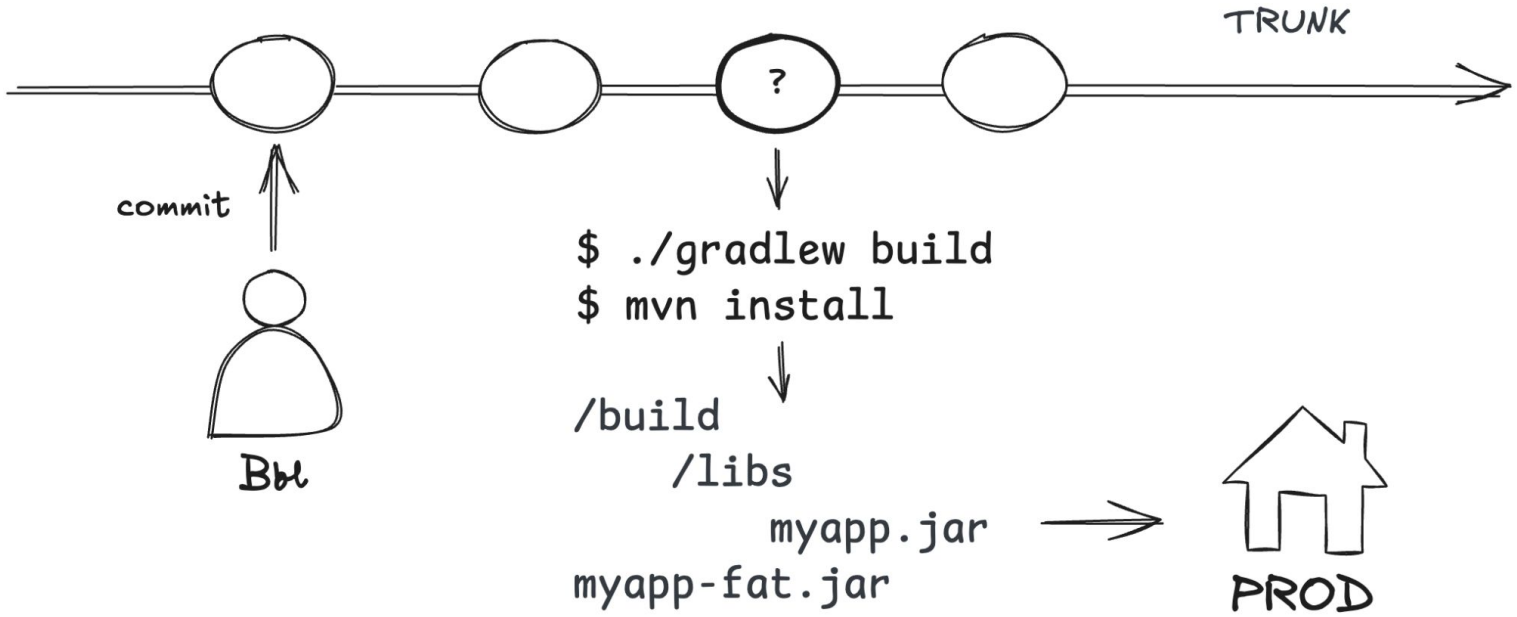
Проблема


TRUNK
?
commit
Вы
$ ./gradlew build
$ mvn install
/build
/libs
myapp.jar
myapp-fat.jar
Версия?

Проблема


TRUNK
?
commit
Вы
$ ./gradlew build
$ mvn install
/build
/libs
myapp.jar
myapp-fat.jar
PROD

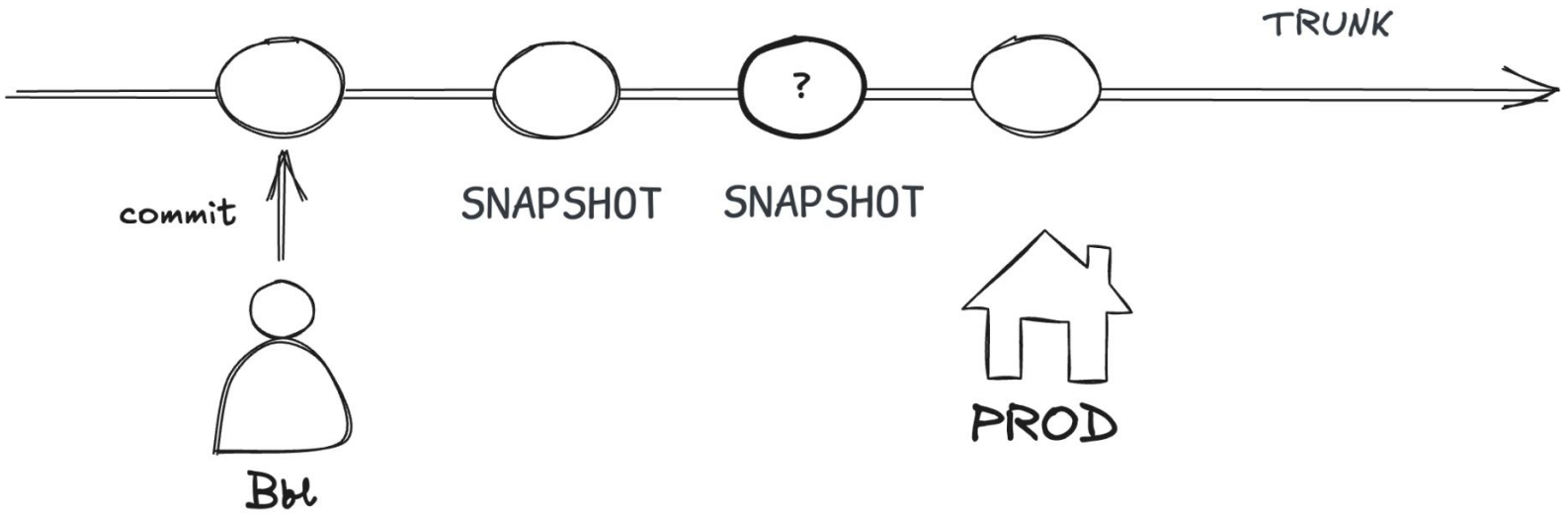
Проблема



Версия?

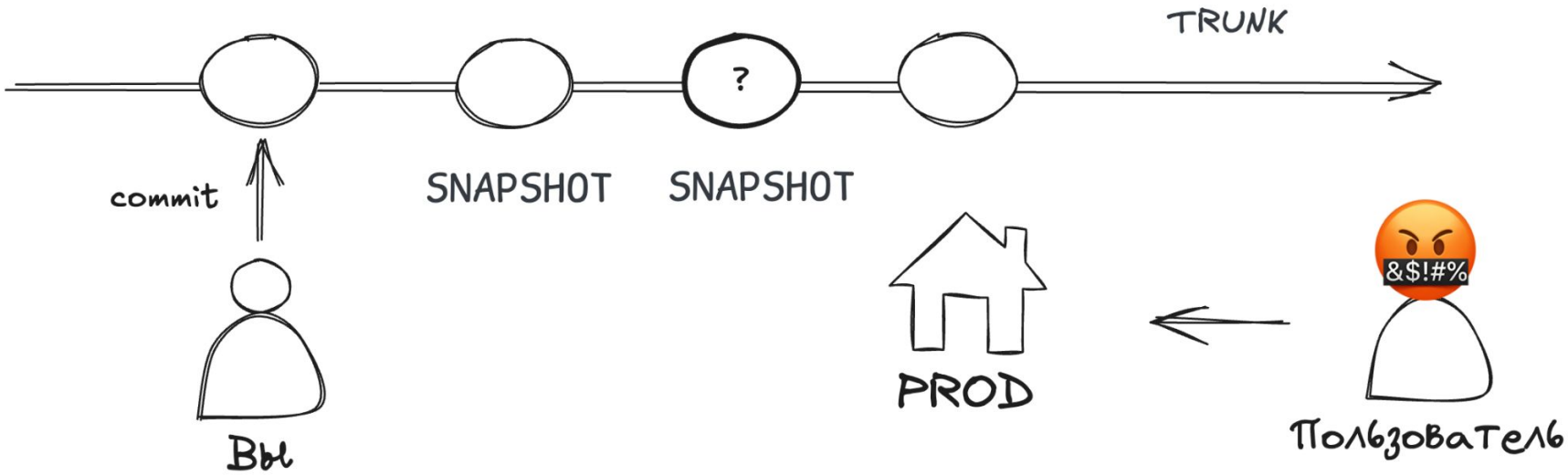
Проблема

```
version=1.0.0-SNAPSHOT
<version>1.0.0-SNAPSHOT</version>
```



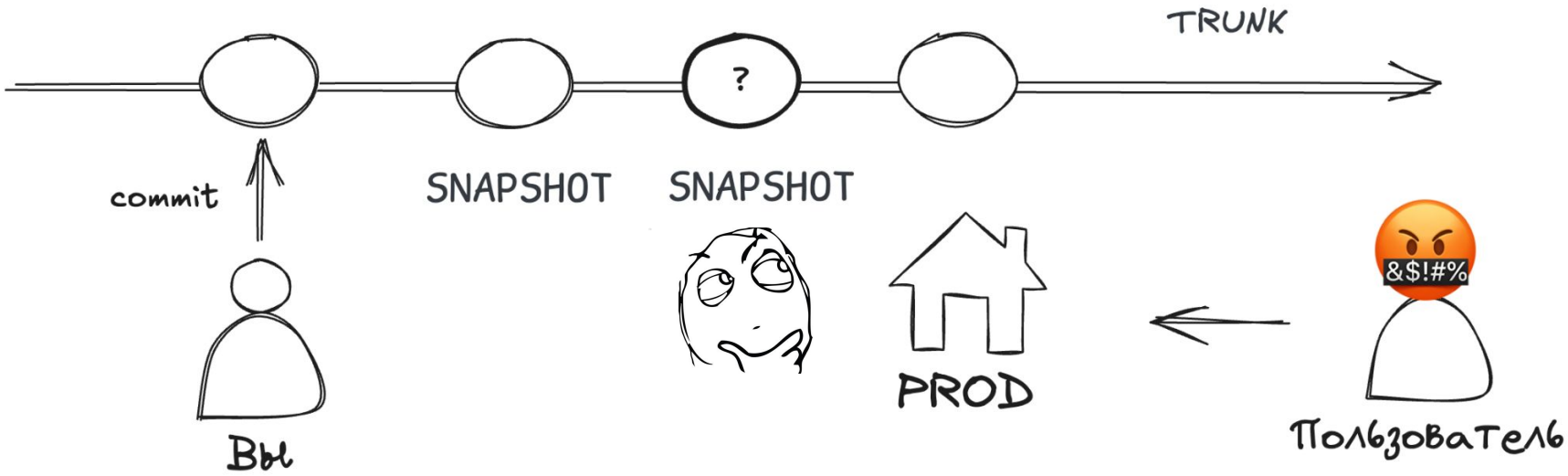
Проблема

```
version=1.0.0-SNAPSHOT
<version>1.0.0-SNAPSHOT</version>
```



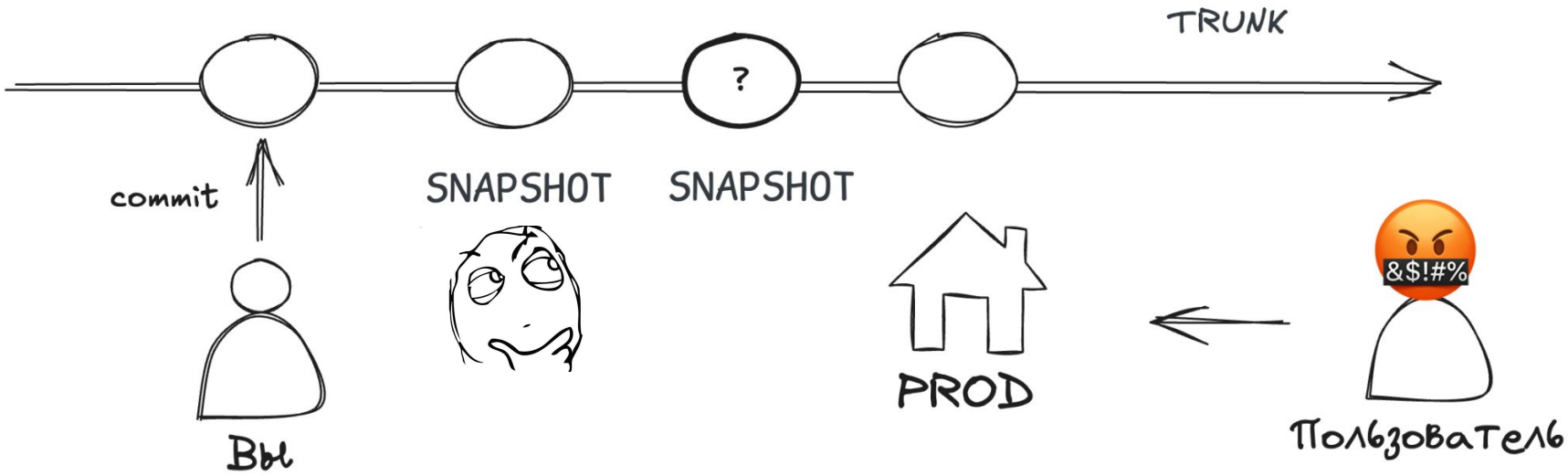
Проблема

```
version=1.0.0-SNAPSHOT
<version>1.0.0-SNAPSHOT</version>
```



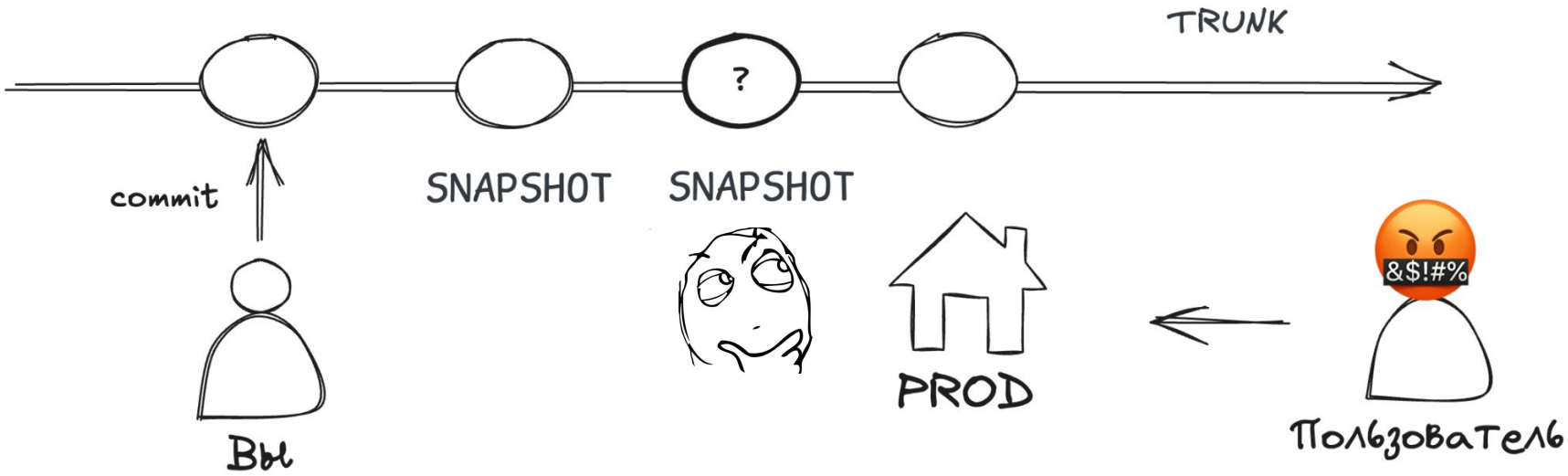
Проблема

```
version=1.0.0-SNAPSHOT
<version>1.0.0-SNAPSHOT</version>
```



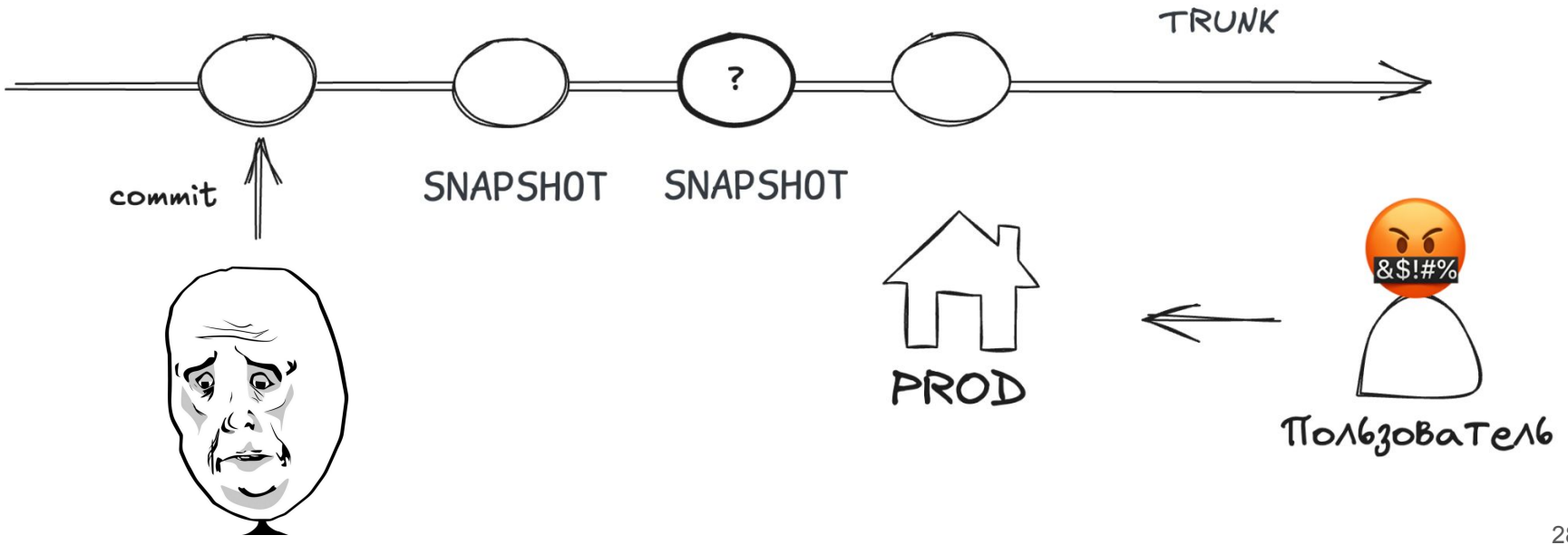
Проблема

```
version=1.0.0-SNAPSHOT
<version>1.0.0-SNAPSHOT</version>
```



Проблема

```
version=1.0.0-SNAPSHOT
<version>1.0.0-SNAPSHOT</version>
```



Commit-id

Use cases

Which version had the bug? Is that deployed already?

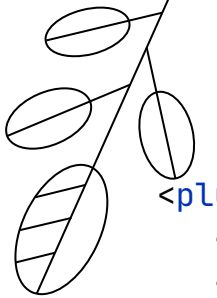
If you develop your maven project inside an git repository you may want to know exactly what changeset is currently deployed. Why is this useful?

I worked in a team where the testers would come up to the development team and say: "hey, feature X is still broken!", to which a dev would reply "But I fixed it this morning!". Then they'd investigate a bit, only to see that the next version which would be deployed very soon included the needed fix, yet the developer already marked it as "ready for testing".

The fix here is obvious: include the version you fixed some bug in the issue comment where you mark it as "ready for testing". You can either do this via smart tooling (recommended), or just manually put in a comment like "fixed in v1.4.3-324-g45xhbghv" (that's a git-describe output - explained in detail bellow), so the testing crew knows it doesn't make sense to pickup testing of this feature until at least "324" (or greater) is included in the version output (it means "number of commits away from the mentioned tag" - readup on git-describe to understand how it works).

https://github.com/git-commit-id

Commit-id



```
<plugin>
    <groupId>io.github.git-commit-id</groupId>
    <artifactId>git-commit-id-maven-plugin</artifactId>
    <version>9.0.1</version>
    <executions>
        <execution>
            <id>get-the-git-infos</id>
            <goals>
                <goal>revision</goal>
            </goals>
            <phase>initialize</phase>
        </execution>
    </executions>
    <configuration>
        <generateGitPropertiesFile>true</generateGitPropertiesFile>
        <generateGitPropertiesFilename>git.properties</generateGitPropertiesFilename>
        <commitIdGenerationMode>full</commitIdGenerationMode>
    </configuration>
</plugin>
```

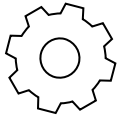
Commit-id

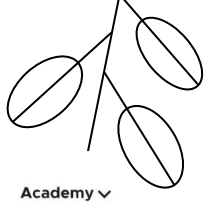
```
public interface Version {
    String TAGS = "${git.tags}";
    String BRANCH = "${git.branch}";
    String DIRTY = "${git.dirty}";
    String REMOTE_ORIGIN_URL = "${git.remote.origin.url}";

    String COMMIT_ID = "${git.commit.id.full}";
}
```

Commit-id

```
Properties properties = new Properties();
properties.load(Version2.class.getClassLoader().getResourceAsStream("git.properties"));
TAGS = String.valueOf(properties.get("git.tags"));
BRANCH = String.valueOf(properties.get("git.branch"));
```

SpringBoot actuator/info





spring

Why Spring Learn Projects Academy

Spring Boot
3.4.4

Search ⌘ + k

- Overview
- Documentation
- Community
- System Requirements
- Installing Spring Boot
- Upgrading Spring Boot
- Tutorials
- Reference
- How-to Guides
  - Spring Boot Application
  - Properties and Configuration
  - Embedded Web Servers
  - Spring MVC
  - Jersey
  - HTTP Clients
  - Logging
  - Data Access

Spring Boot / How-to Guides / Build

Generate Git Information

Both Maven and Gradle allow generating a `git.properties` file containing information about the state of your `git` source code repository when the project was built.

For Maven users, the `spring-boot-starter-parent` POM includes a pre-configured plugin to generate a `git.properties` file. To use it, add the following declaration for the `Git Commit Id Plugin` to your POM:

XML

```xml
<build>
    <plugins>
        <plugin>
            <groupId>io.github.git-commit-id</groupId>
            <artifactId>git-commit-id-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

Gradle users can achieve the same result by using the `gradle-git-properties` plugin, as shown in the following example:

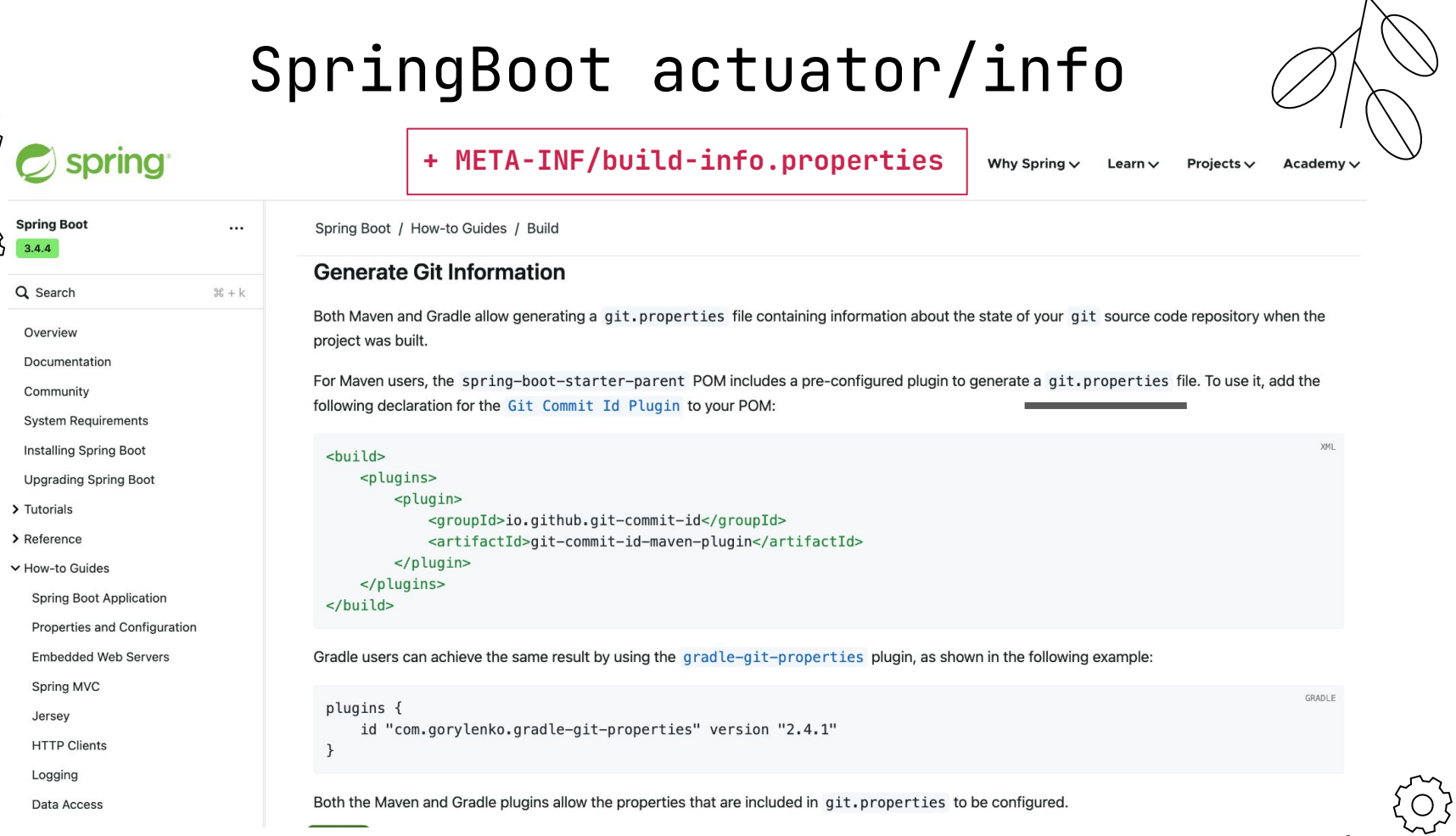
GRADLE

```
plugins {
    id "com.gorylenko.gradle-git-properties" version "2.4.1"
}
```

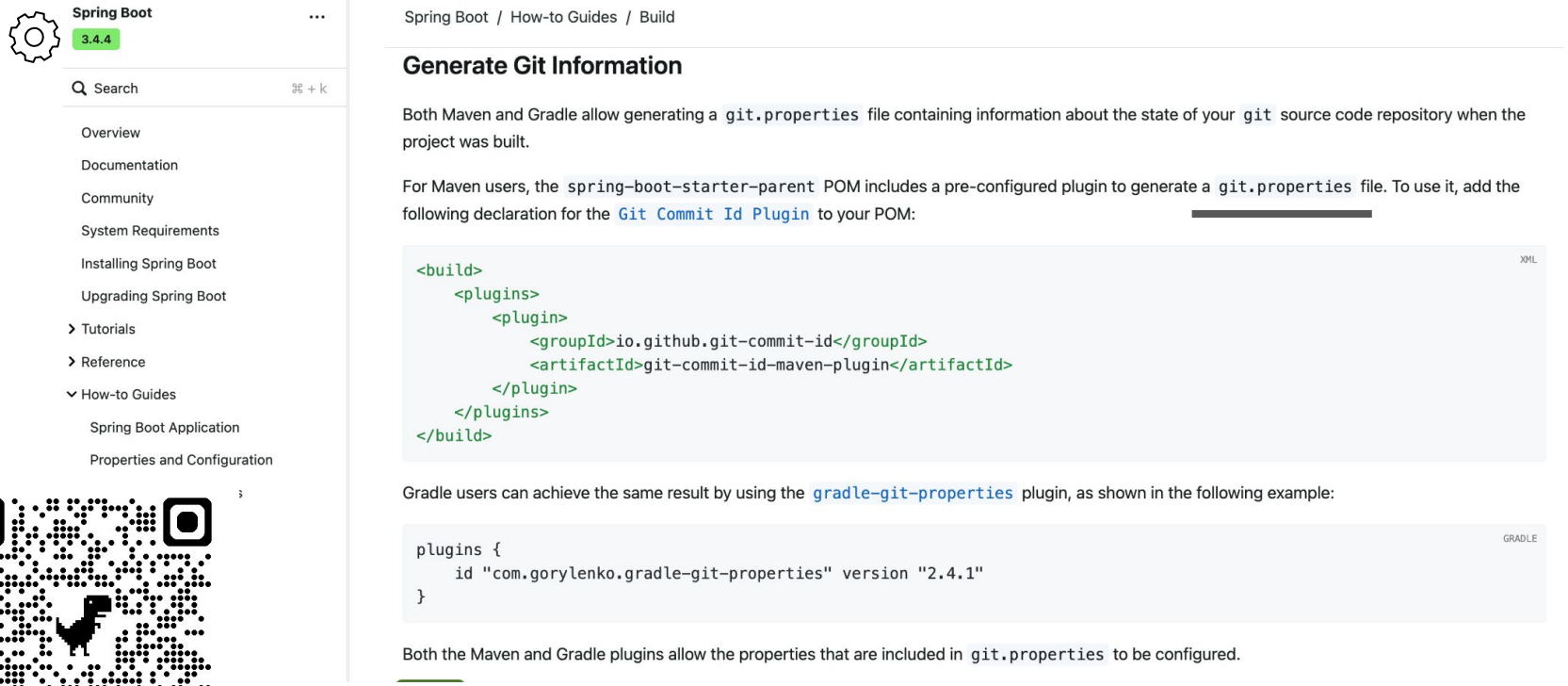
Both the Maven and Gradle plugins allow the properties that are included in `git.properties` to be configured.

SpringBoot actuator/info

+ META-INF/build-info.properties

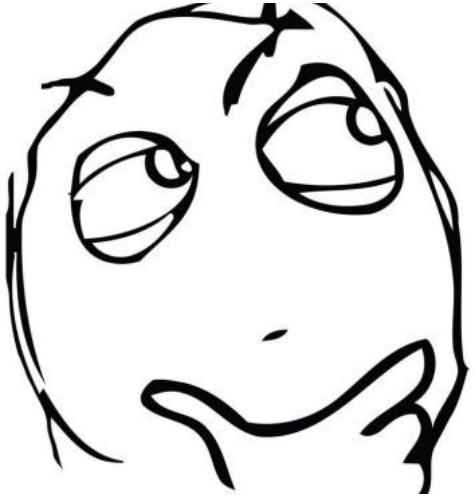


SpringBoot actuator/info



https://github.com/n0mer/gradle-git-properties

Что если просто именовать артефакты правильно с версиями?



релиз.jar

релиз фикс.jar

релиз 3.jar

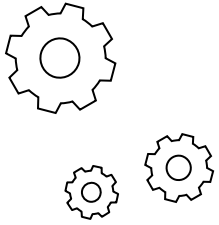
релиз QA.jar

релиз copy.jar

релиз финальный.jar

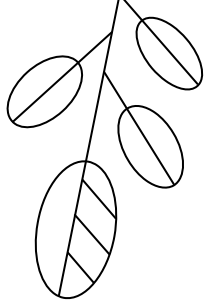
релиз final 2.jar

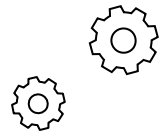
релиз последний в рот оно.jar

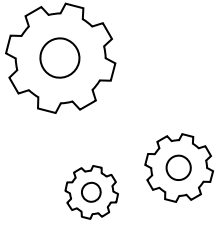


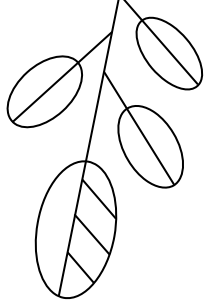
MAJOR.MINOR.PATCH

semver.org







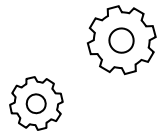


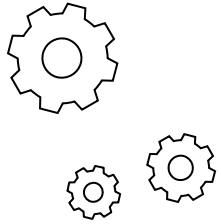
MAJOR.MINOR.PATCH

semver.org

YYYY.MM.MICRO

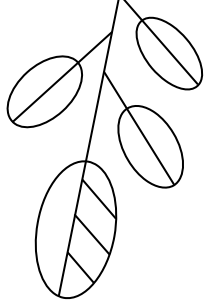
calver.org





MAJOR.MINOR.PATCH

semver.org

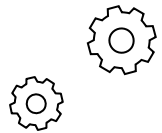


YYYY.MM.MICRO

calver.org

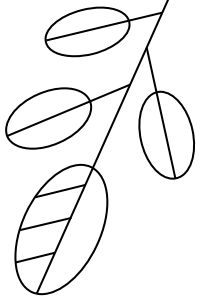
0.MINOR.PATCH

0ver.org

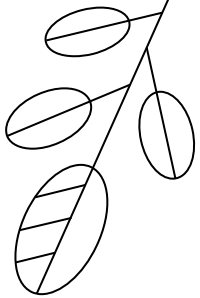


СломВер 3.0: МНОГО.МАЛО.ЧУТОЧКУ

Владимир Ситников
vladimirsitnikv

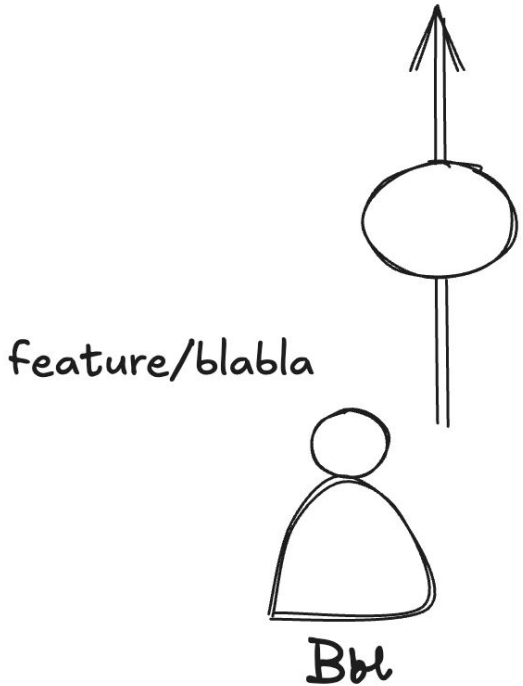


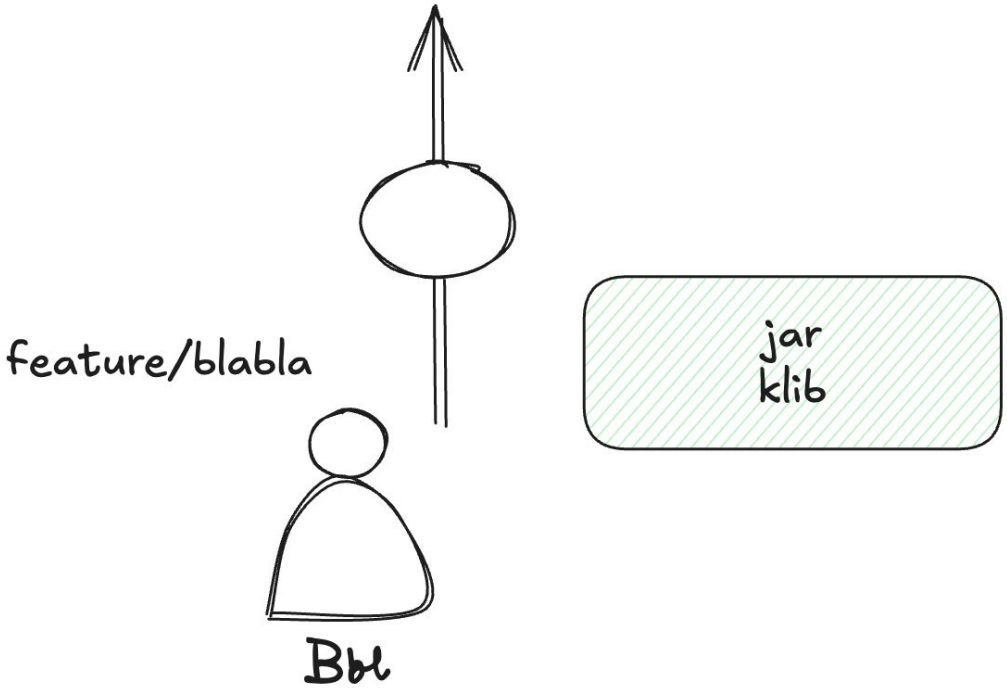
Итак, мы открыли для себя и git и SemVer



Для бэка на Java есть куча этапов, где версионирование **критически** необходимо.

Особенно, если это публичный проект.


feature/blabla
Вы


feature/blabla
jar
klib
Вы

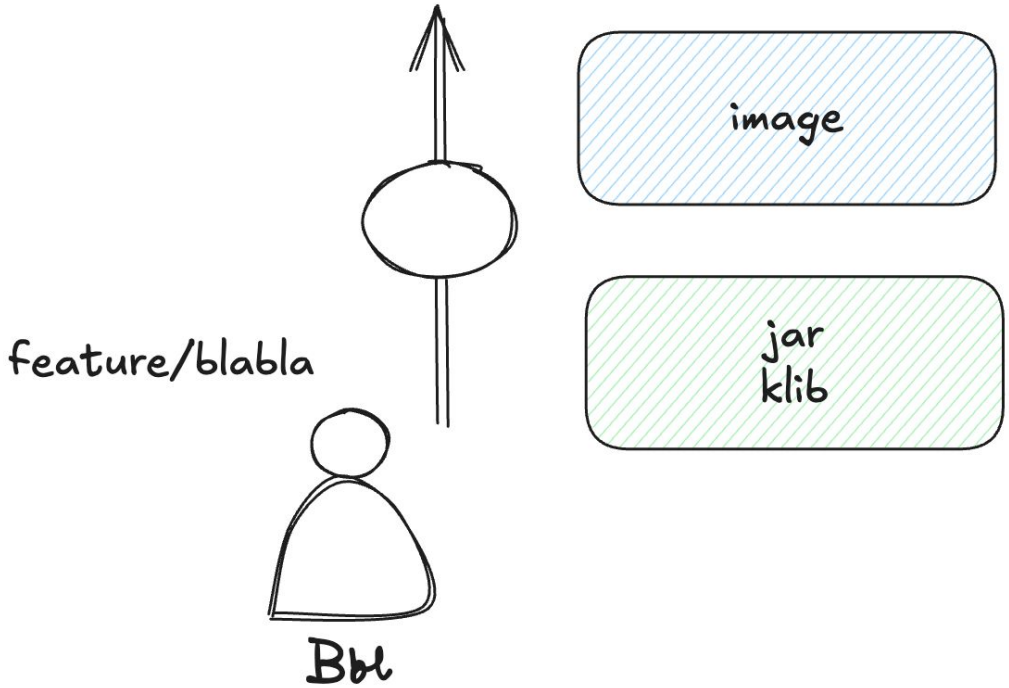
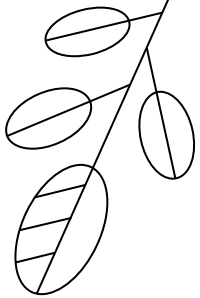
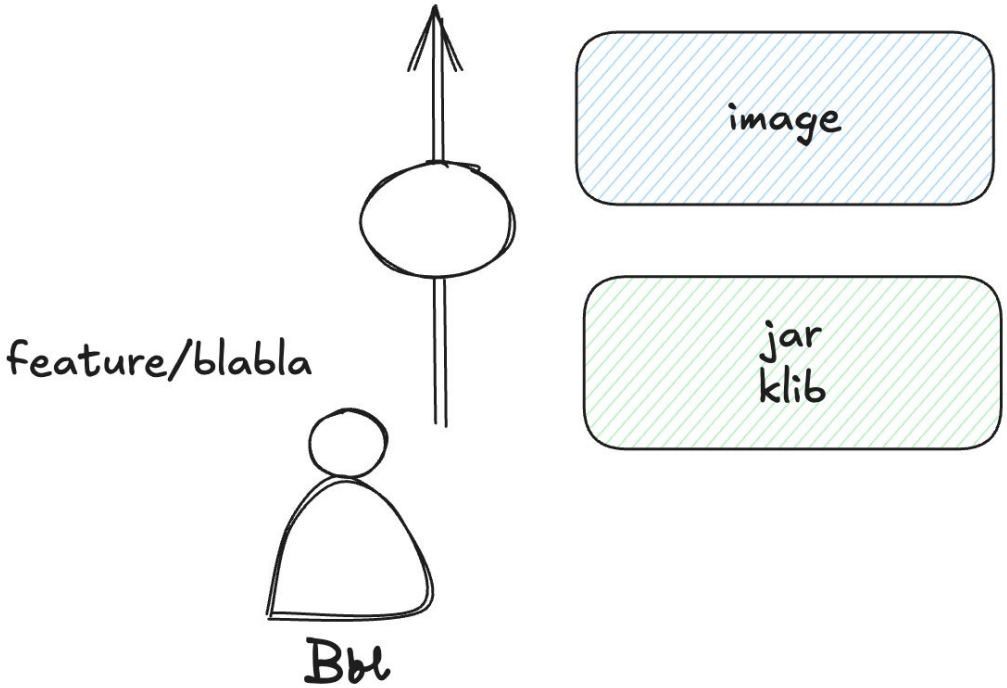
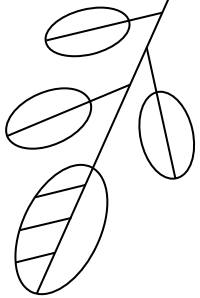


image
jar
klib
feature/blabla
Вы

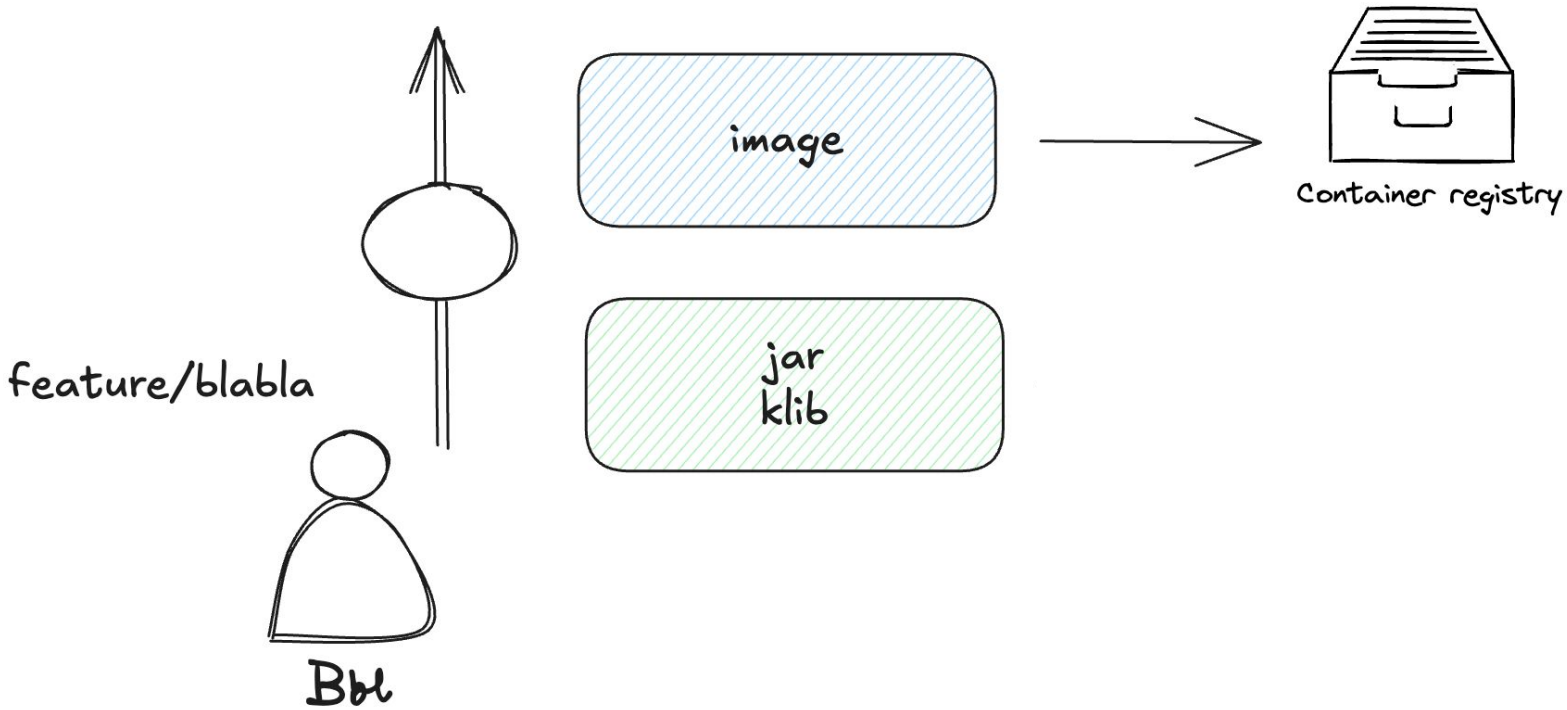


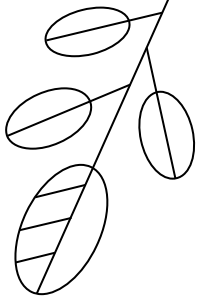
```
./gradlew bootBuildImage --imageName=...
./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=
```





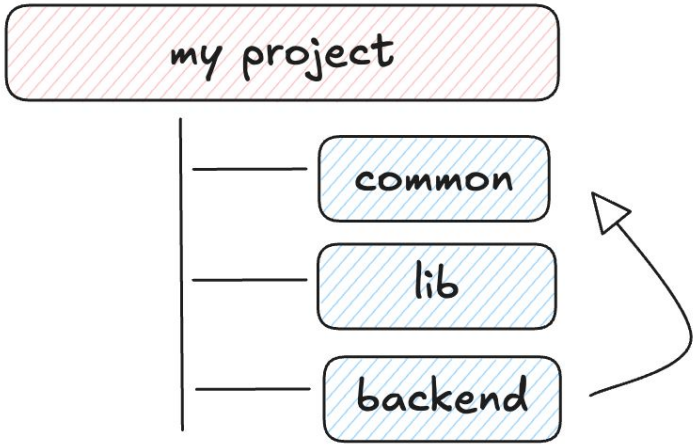
```
./gradlew bootBuildImage --imageName=...
./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=
```



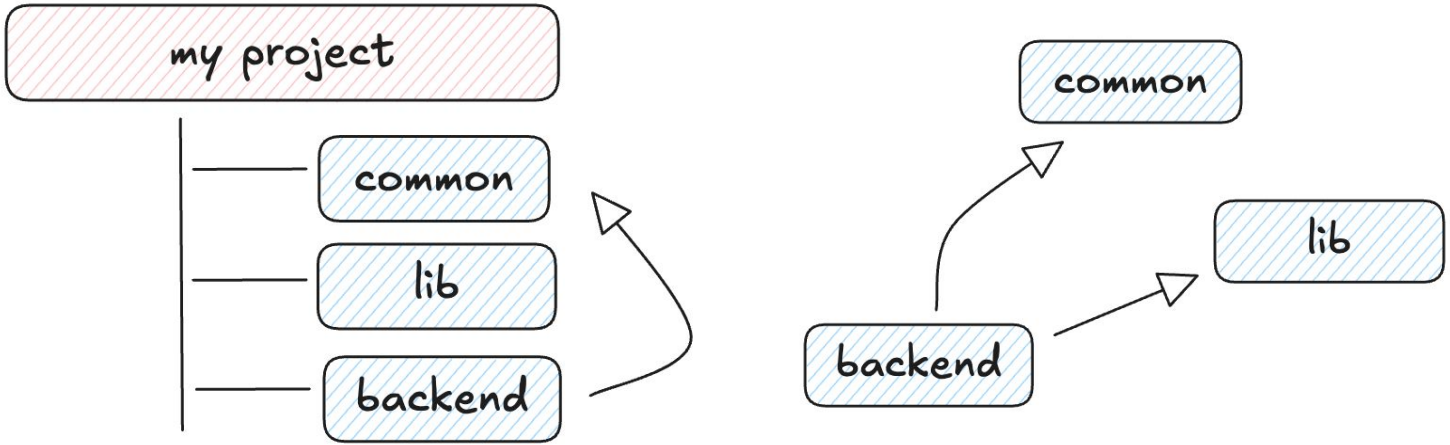


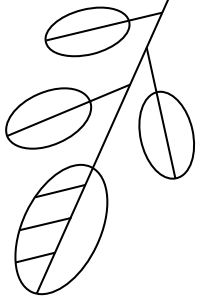
А если **еще** мульти-репозиторий или библиотека

А если **еще** мульти-репозиторий или библиотека

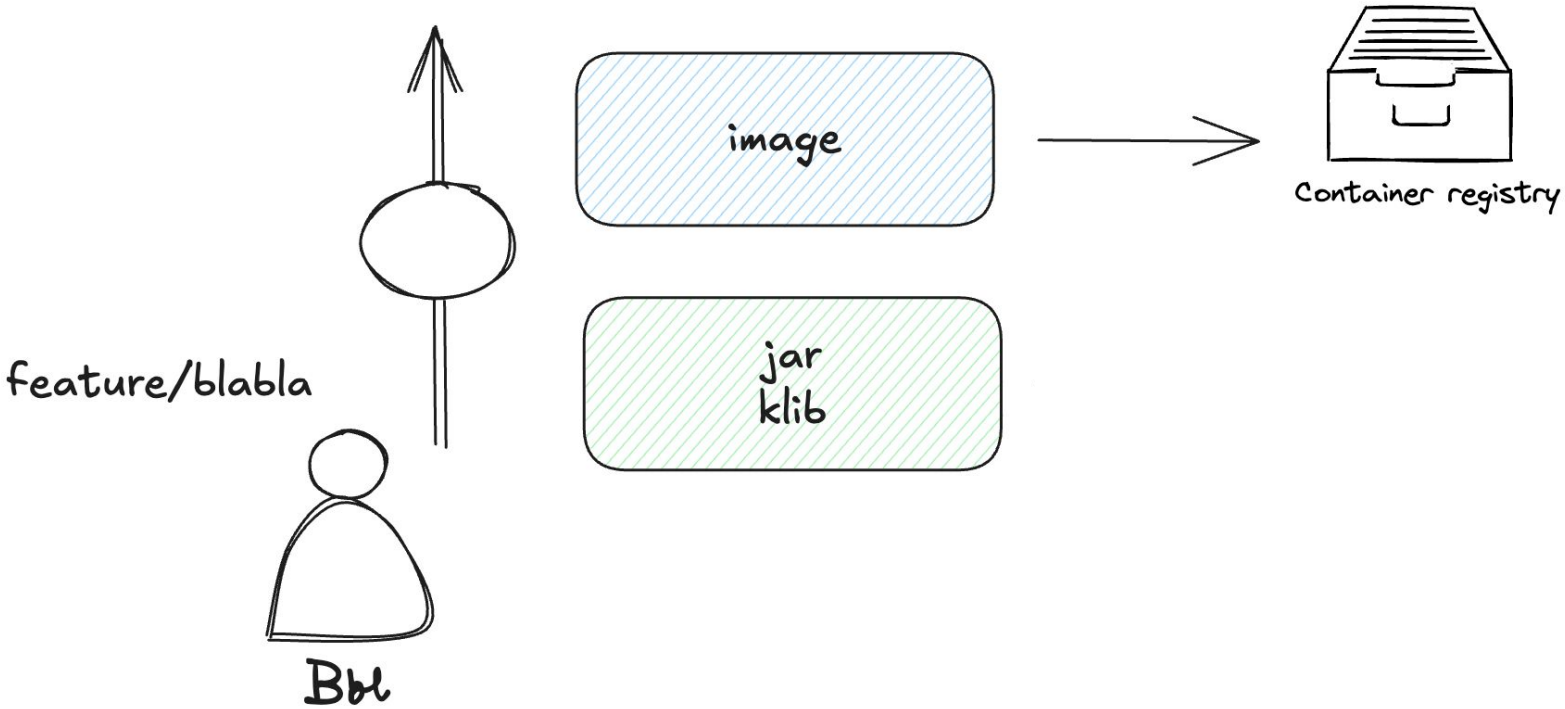


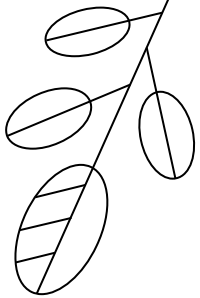
А если **еще** мульти-репозиторий или библиотека



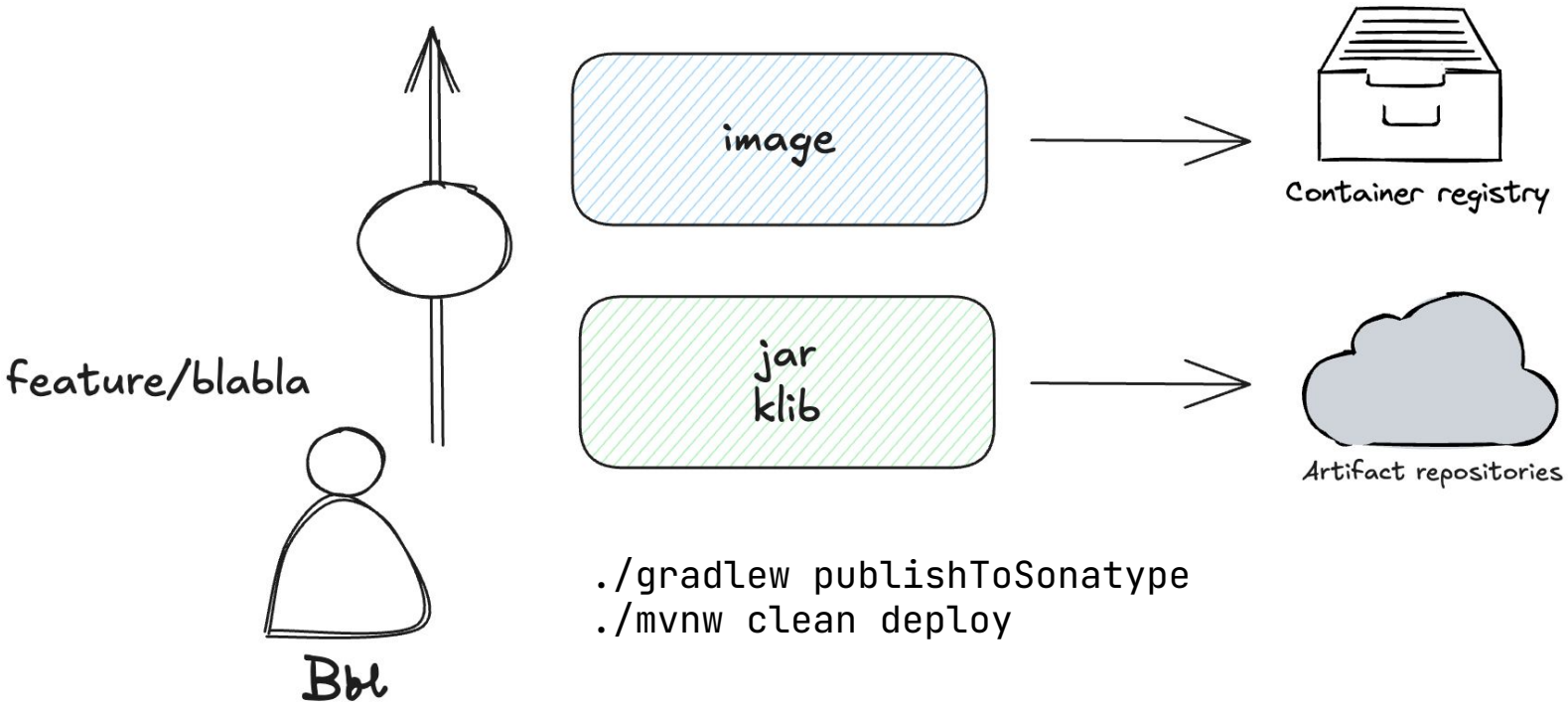


```
./gradlew bootBuildImage --imageName=...
./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=
```

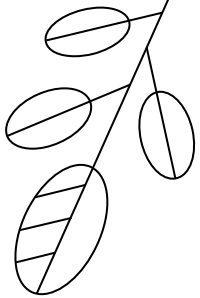




```
./gradlew bootBuildImage --imageName=...
./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=
```

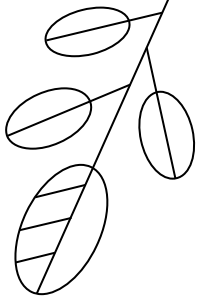



```
./gradlew publishToSonatype
./mvnw clean deploy
```



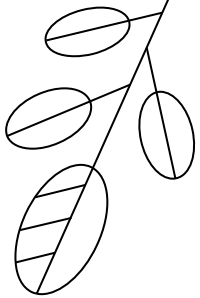
```
./gradlew bootBuildImage --imageName=...
./mvnw spring-boot:build-image -Dspring-boot.build-image.imageName=
```


```
./gradlew publishToSonatype
./mvnw clean deploy
```



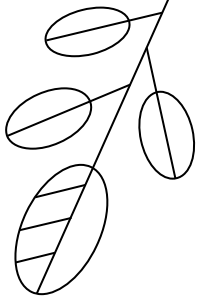
Gradle как пример

```
tasks.named<BootBuildImage>("bootBuildImage") {
    imageName =
      "ghcr.io/name/${project.name}:${project.versionForDockerImages()}"
    isPublish = true
    docker {
        publishRegistry {
            username = providers.environmentVariable("GHCR_USER")
            password = providers.environmentVariable("GHCR_PWD")
            url = "https://ghcr.io"
        }
    }
}
```



Gradle как пример

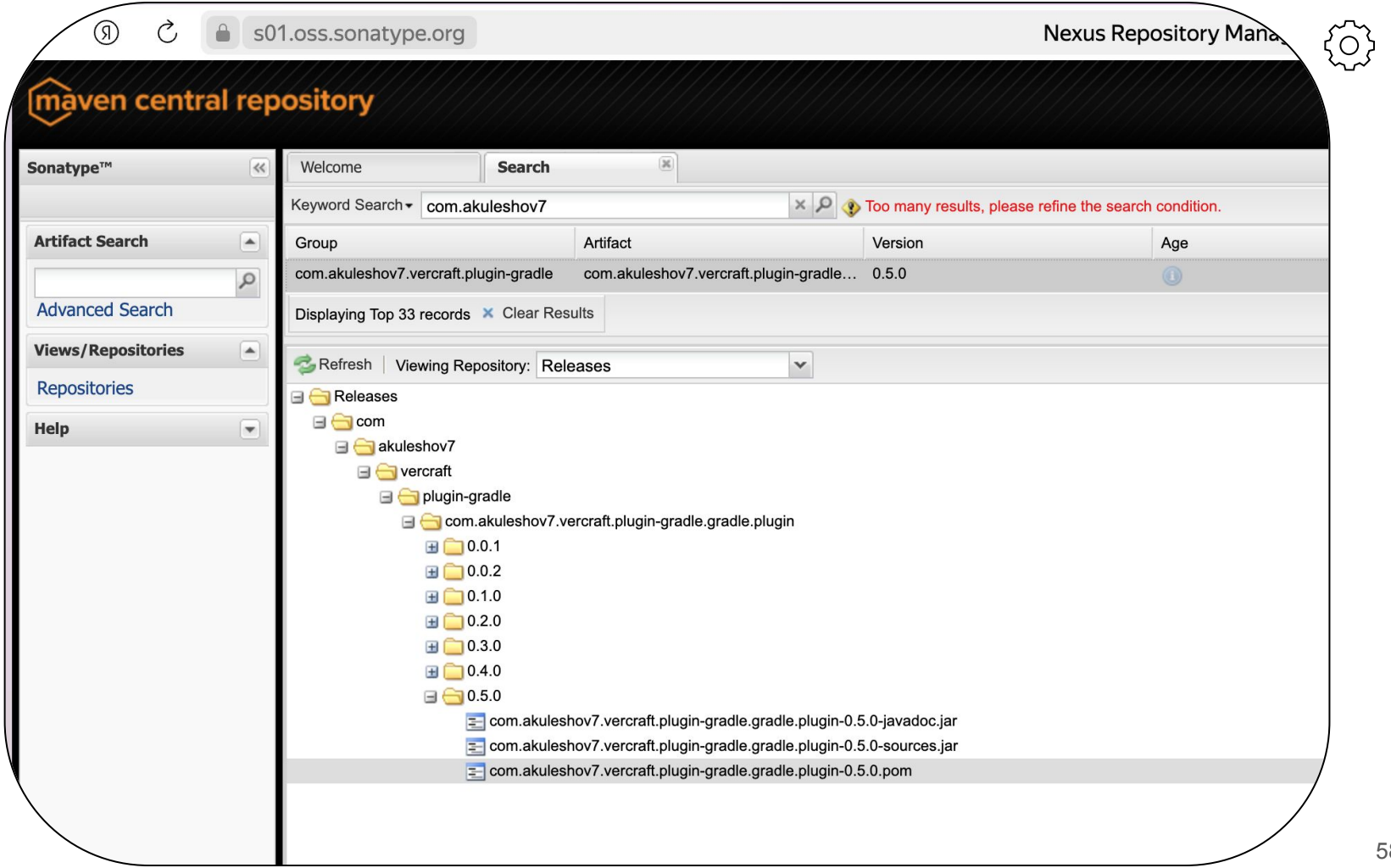
```
internal fun Project.versionForDockerImages(): String =
    (project.findProperty("build.dockerTag") as String?
        ?: project.version.toString())
        .replace(Regex("[^._\\-a-zA-Z0-9]"), "-")
```

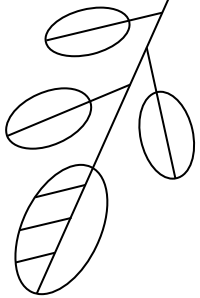


Gradle как пример

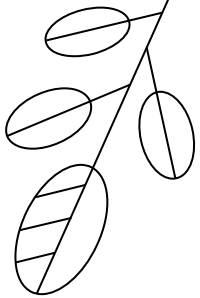
```
publishing {
    publications {
        create<MavenPublication>("maven") {
            from(components["java"])
        }
    }
}
```

myproject-X.X.X.jar


s01.oss.sonatype.org
Nexus Repository Mana
maven central repository
Sonatype™
Artifact Search
Advanced Search
Views/Repositories
Repositories
Help
Welcome
Search
Keyword Search
com.akuleshov7
Too many results, please refine the search condition.
Group
Artifact
Version
Age
com.akuleshov7.vercraft.plugin-gradle
com.akuleshov7.vercraft.plugin-gradle…
0.5.0
Displaying Top 33 records
Clear Results
Refresh
Viewing Repository:
Releases
Releases
com
akuleshov7
vercraft
plugin-gradle
com.akuleshov7.vercraft.plugin-gradle.gradle.plugin
0.0.1
0.0.2
0.1.0
0.2.0
0.3.0
0.4.0
0.5.0
com.akuleshov7.vercraft.plugin-gradle.gradle.plugin-0.5.0-javadoc.jar
com.akuleshov7.vercraft.plugin-gradle.gradle.plugin-0.5.0-sources.jar
com.akuleshov7.vercraft.plugin-gradle.gradle.plugin-0.5.0.pom



Как это версионирование автоматизировать?

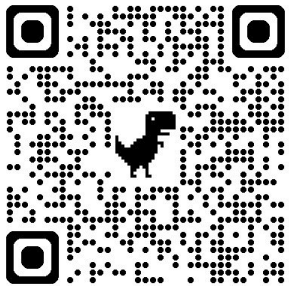


Jgitver

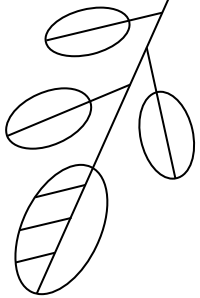
Mvn & Gradle plugin support

```
id("fr.brouillard.oss.gradle.jgitver") version("0.9.1")
```

```
$ ./gradlew version
>> Version: 0.5.1-2-main
```



https://github.com/jgitver/jgitver



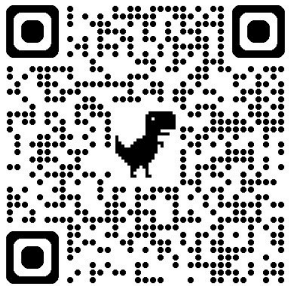
Jgitver

Mvn & Gradle plugin support

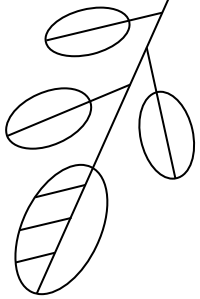
```
id("fr.brouillard.oss.gradle.jgitver") version("0.9.1")
```

```
$ ./gradlew version
>> Version: 0.5.1-2-main
```

Всё на тегах



https://github.com/jgitver/jgitver



Andrew Oberstar

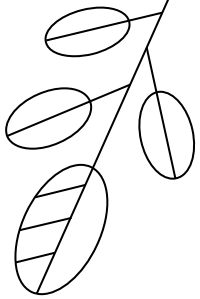
Project Status

gradle-git has been around since 2012 and has evolved quite a bit from the original release. In order to continue to evolve these features, this project is being broken up into multiple repositories. As such:

- gradle-git will no longer be maintained

feature	replacement	comments
`org.ajoberstar.grgit`	grgit	Grgit has been an independent project since 2013 and has been stable for quite a while. Version 2.0 removed some deprecated features, but otherwise is fully compatible with existing usage. It also integrates the `org.ajoberstar.grgit` plugin directly into the project.
`org.ajoberstar.github-pages`	gradle-git-publish	`org.ajoberstar.git-publish` is a more robust version of the old plugin. It is functionally equivalent (or better), but does require porting configuration over as noted in the README.
`org.ajoberstar.release-*`	reckon	Reckon focuses solely on determining your project version (and assisting with tagging and pushing that tag). It provides an opinionated model of how to apply semantic versioning, with more finite configuration options.
`org.ajoberstar.release-*`	nebula-release	If reckon doesn't suit your needs, nebula-release has forked the gradle-git release plugin and can serve as a replacement for this.

https://github.com/ajoberstar/gradle-git



Andrew Oberstar

Project Status

gradle-git has been around since 2012 and has evolved quite a bit from the original release. In order to continue to evolve these features, this project is being broken up into multiple repositories. As such:

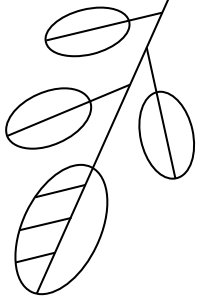
- gradle-git will no longer be maintained

feature	replacement	comments
`org.ajoberstar.grgit`	grgit	Grgit has been an independent project since 2013 and has been stable for quite a while. Version 2.0 removed some deprecated features, but otherwise is fully compatible with existing usage. It also integrates the `org.ajoberstar.grgit` plugin directly into the project.
`org.ajoberstar.github-pages`	gradle-git-publish	`org.ajoberstar.git-publish` is a more robust version of the old plugin. It is functionally equivalent (or better), but does require porting configuration over as noted in the README.
`org.ajoberstar.release-*`	reckon	Reckon focuses solely on determining your project version (and assisting with tagging and pushing that tag). It provides an opinionated model of how to apply semantic versioning, with more finite configuration options.
`org.ajoberstar.release-*`	nebula-release	If reckon doesn't suit your needs, nebula-release has forked the gradle-git release plugin and can serve as a replacement for this.

https://github.com/ajoberstar/gradle-git

Nebula

https://nebula-plugins.github.io/



nebula-release-plugin

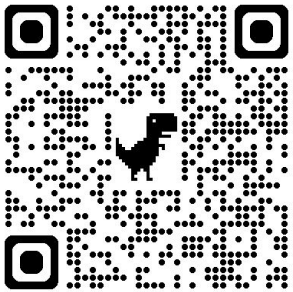
```
nebulaRelease {
  Set<String> releaseBranchPatterns = [/main/, /(release(-|\/))?\d+(\.\d+)?\.x/, /v?\d+\.\d+\.\d+/] as Set
  Set<String> excludeBranchPatterns = [] as Set
  String shortenedBranchPattern = /(?:(?:bugfix|feature|hotfix|release)(?:-|\/))?(.+)/

  void addReleaseBranchPattern(String pattern)
  void addExcludeBranchPattern(String pattern)
}
```

https://nebula-plugins.github.io/

Reckon

settings.gradle.kts:

```
plugins {
   id("org.ajoberstar.reckon.settings") version "0.19.1"
}
```

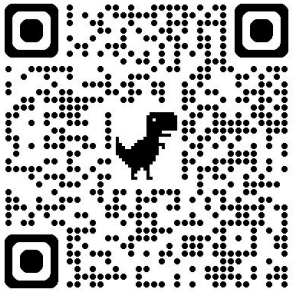


https://github.com/ajoberstar/reckon

Reckon

settings.gradle.kts:

```
plugins {
   id("org.ajoberstar.reckon.settings") version "0.19.1"
}

extensions.configure<org.ajoberstar.reckon.gradle.ReckonExtension> {
   setDefaultInferredScope("patch")
   stages("alpha", "final")
   setScopeCalc(calcScopeFromProp().or(calcScopeFromCommitMessages()))
   setStageCalc(calcStageFromProp())
}
```

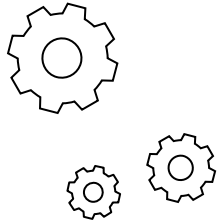


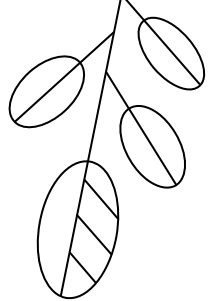
https://github.com/ajoberstar/reckon

Reckon

```
fun Project.configureVersioning() {
    apply<ReckonPlugin>()

    val isSnapshot = hasProperty("reckon.stage")
                     && property("reckon.stage") == "snapshot"
    configure<ReckonExtension> {
        setDefaultInferredScope("patch")
        if (isSnapshot) {
            snapshots()
        } else {
            stages("alpha", "rc", "final")
        }
          setScopeCalc(calcScopeFromProp().or(calcScopeFromCommitMessages()))
          setStageCalc(calcStageFromProp())
    }
}
```

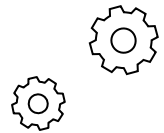
Reckon





Reckoned version: **0.5.1**-**alpha**.0.2+20250402T095622Z

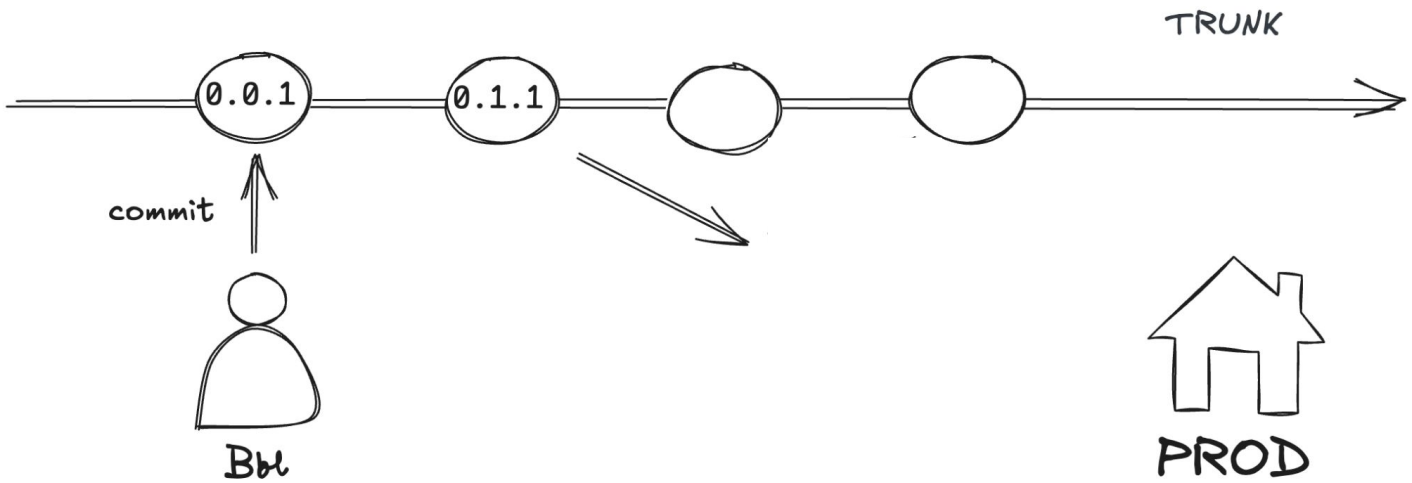
Reckoned version: 0.6.0



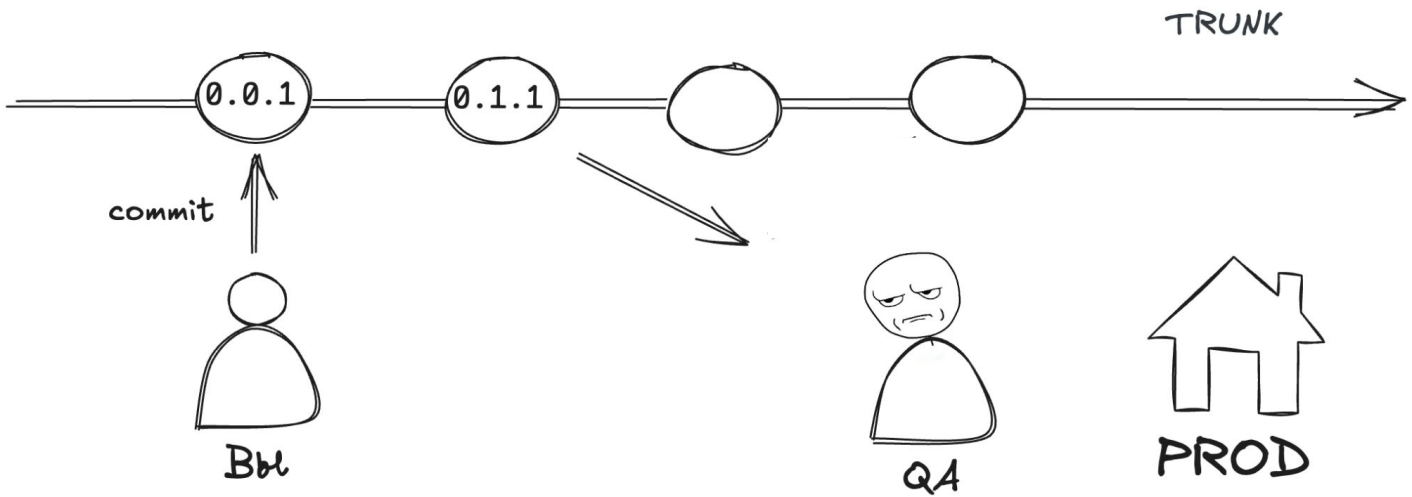
Reckon

```
./gradlew reckonTagPush
```

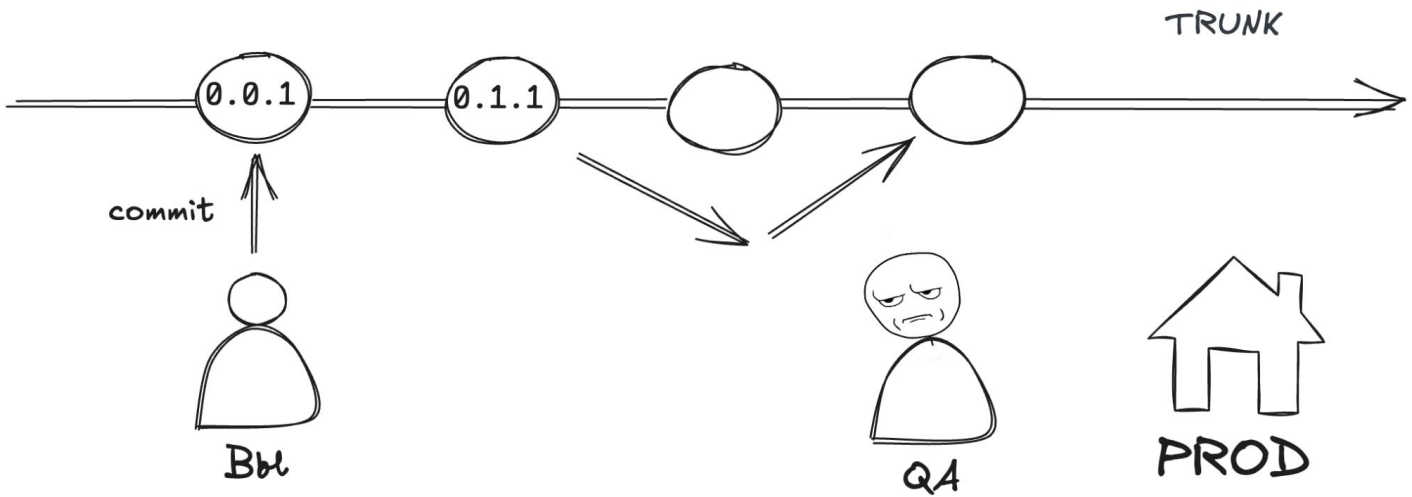
Реальность



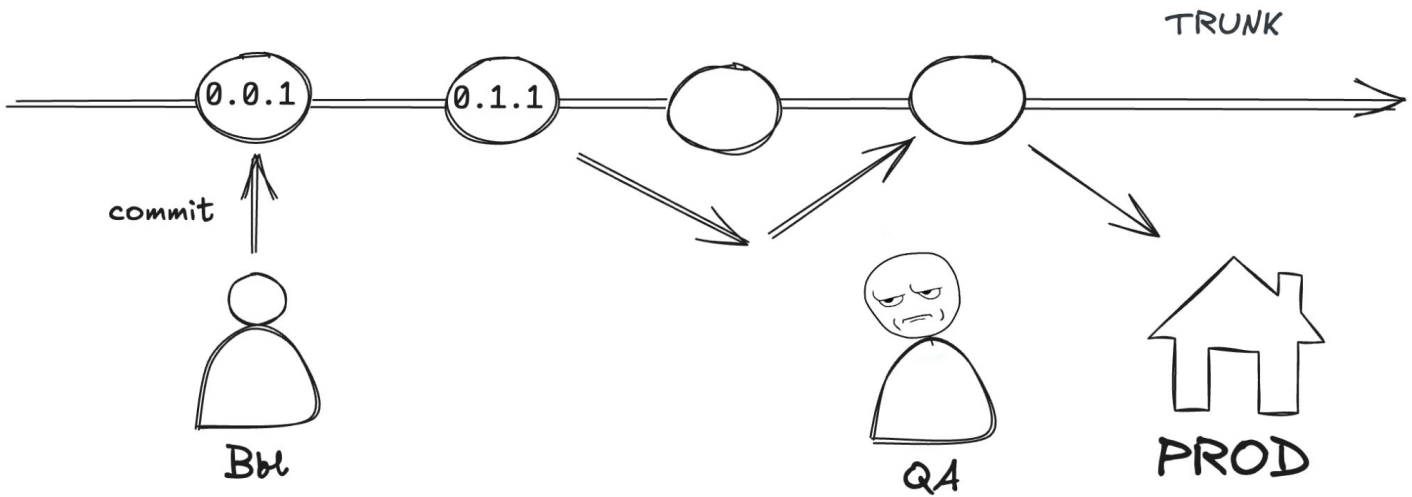
Реальность

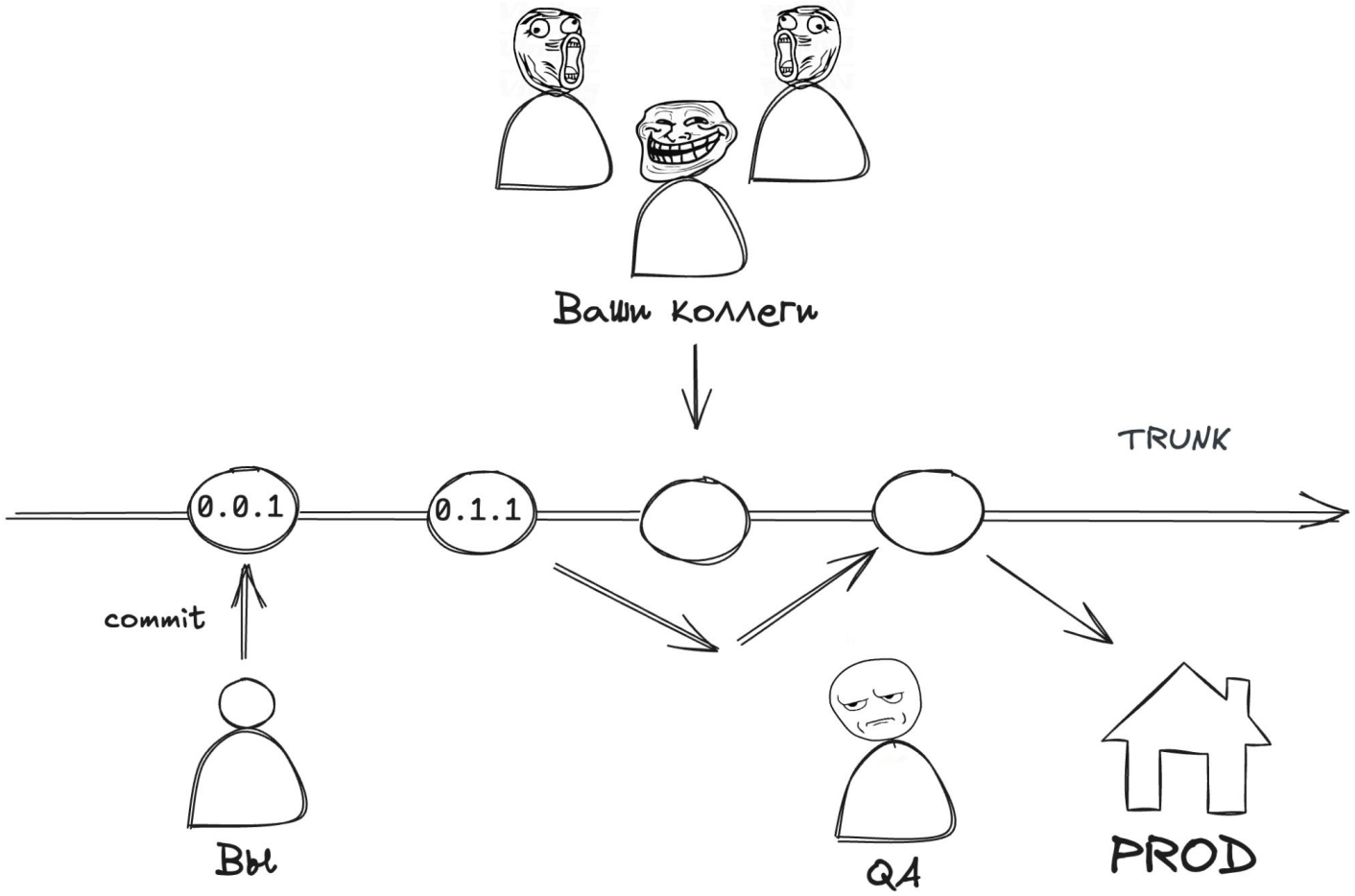


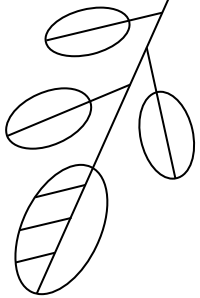
Реальность



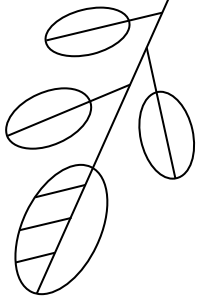
Реальность



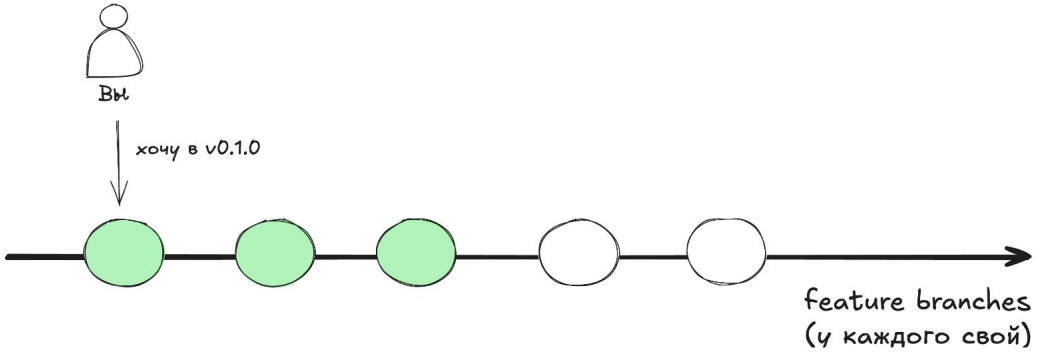

Ваши коллеги
TRUNK
0.0.1
0.1.1
commit
Вы
QA
PROD

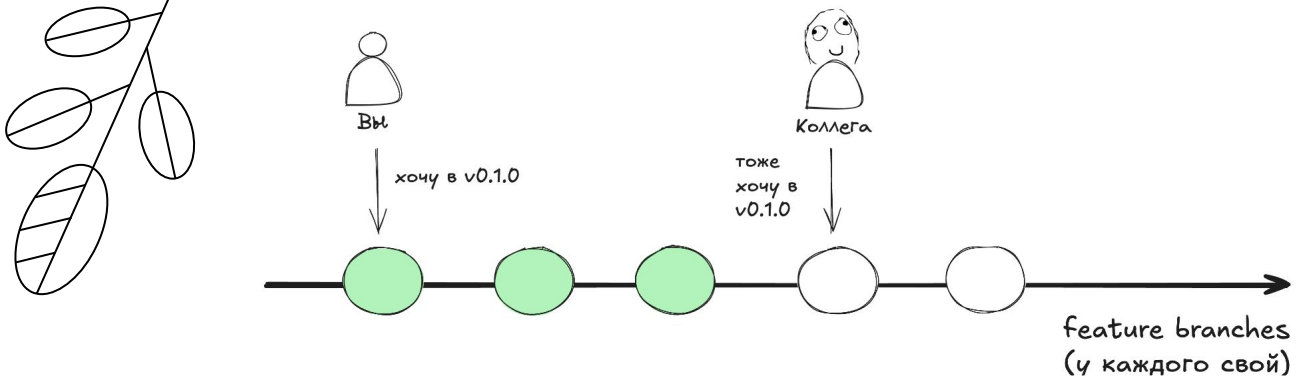


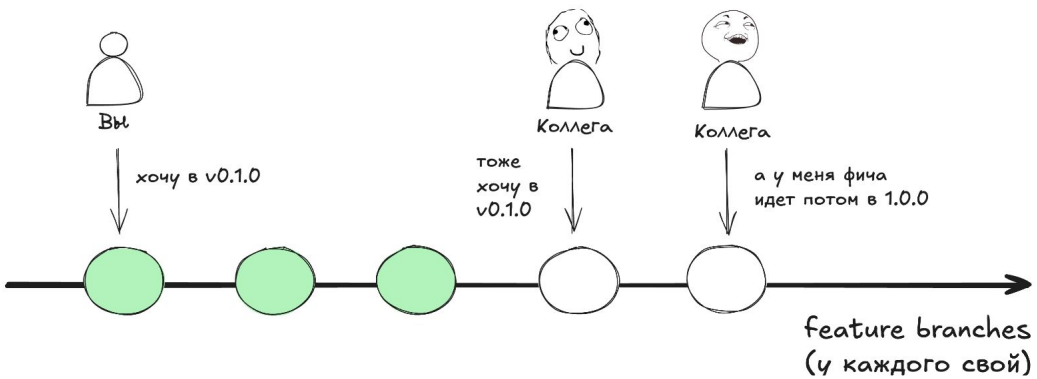
Выбор модели версионирования и ее автоматизация напрямую зависят от **модели бранчевания** и **релизной модели**

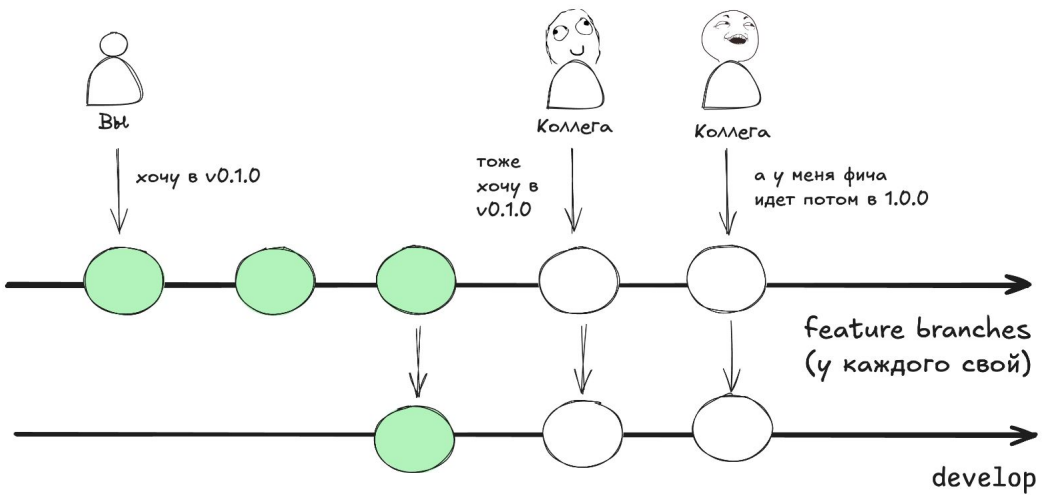


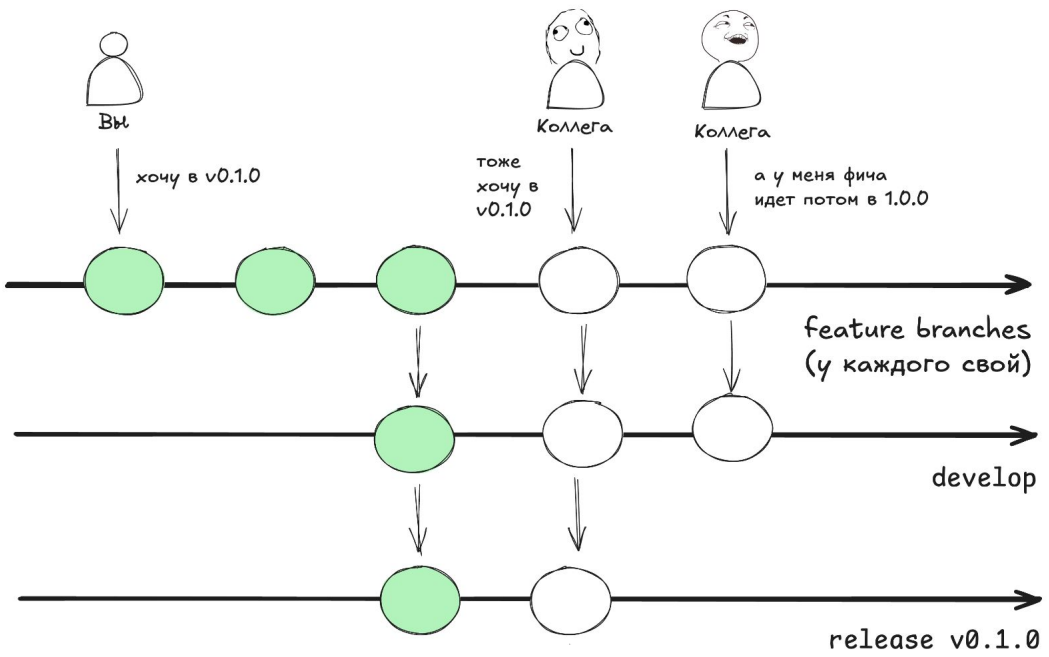
Git Flow

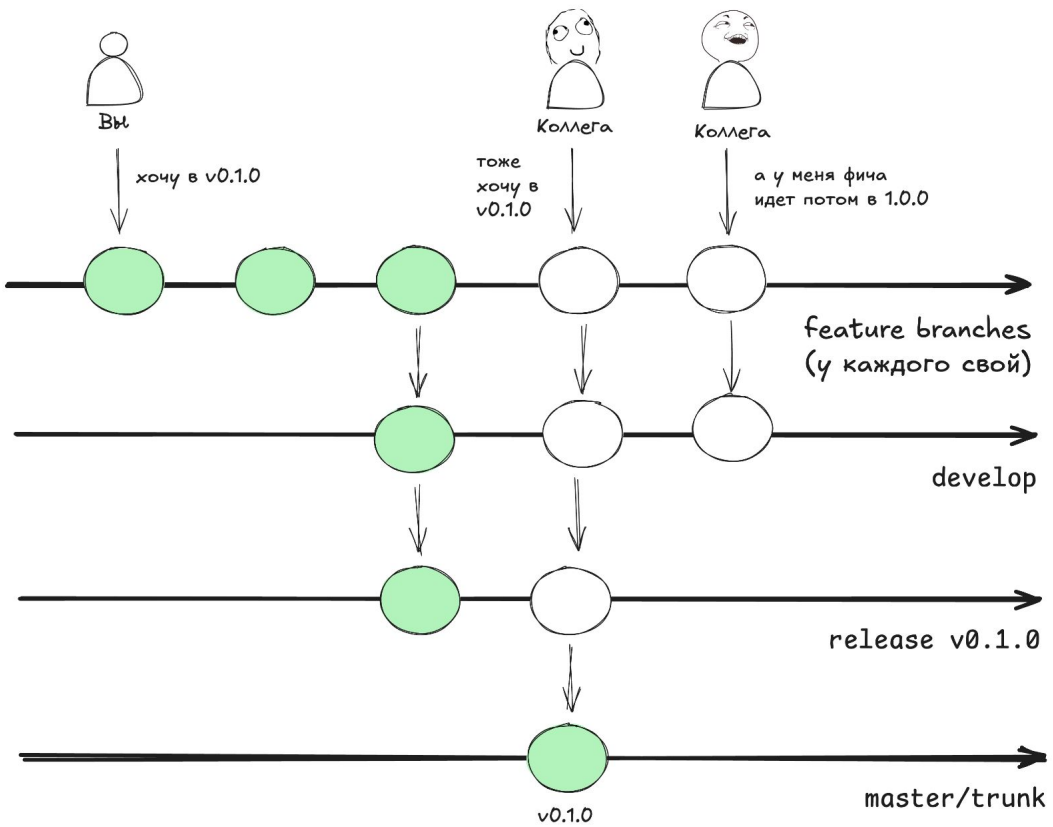

Вы
хочу в v0.1.0
feature branches
(у каждого свой)

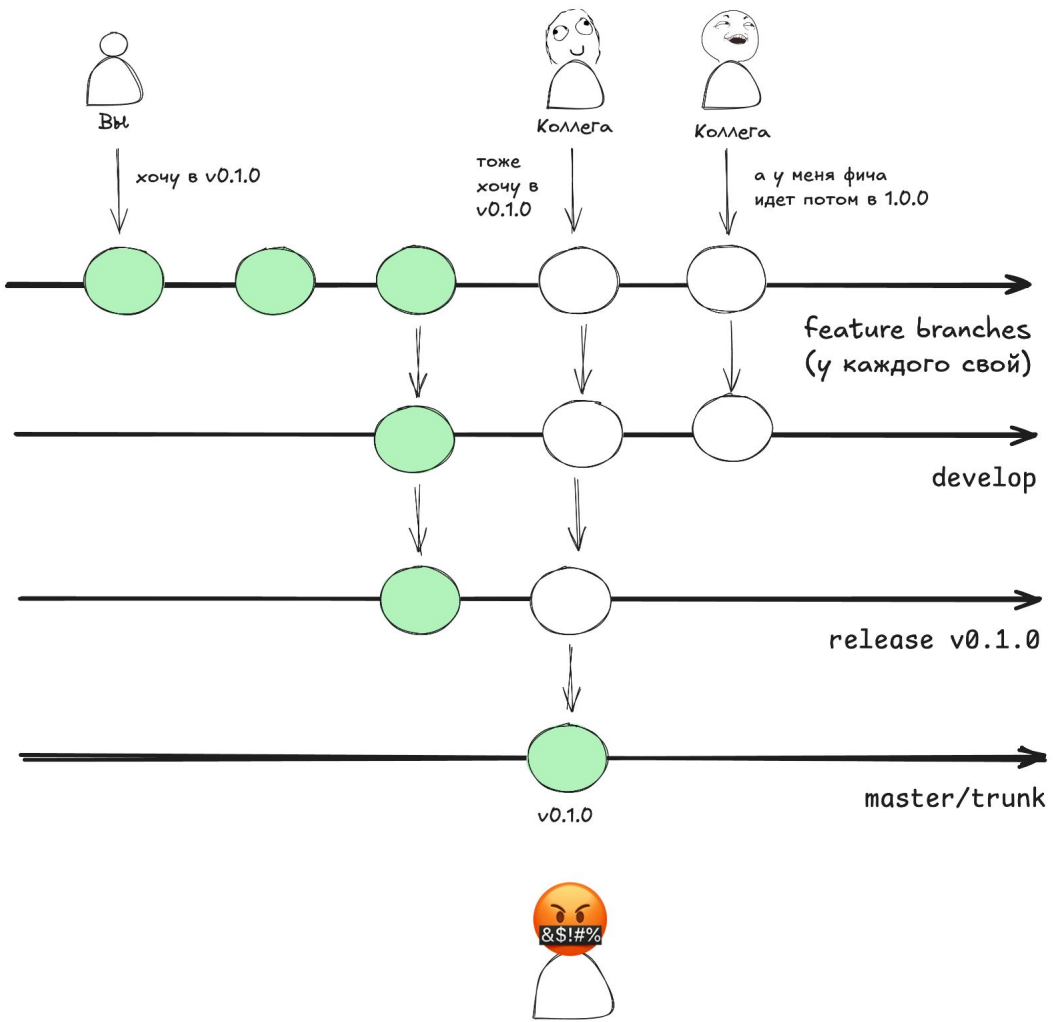

Вы
Коллега
хочу в v0.1.0
тоже
хочу в
v0.1.0
feature branches
(у каждого свой)


Вы
Коллега
Коллега
хочу в v0.1.0
тоже
хочу в
v0.1.0
а у меня фича
идет потом в 1.0.0
feature branches
(у каждого свой)

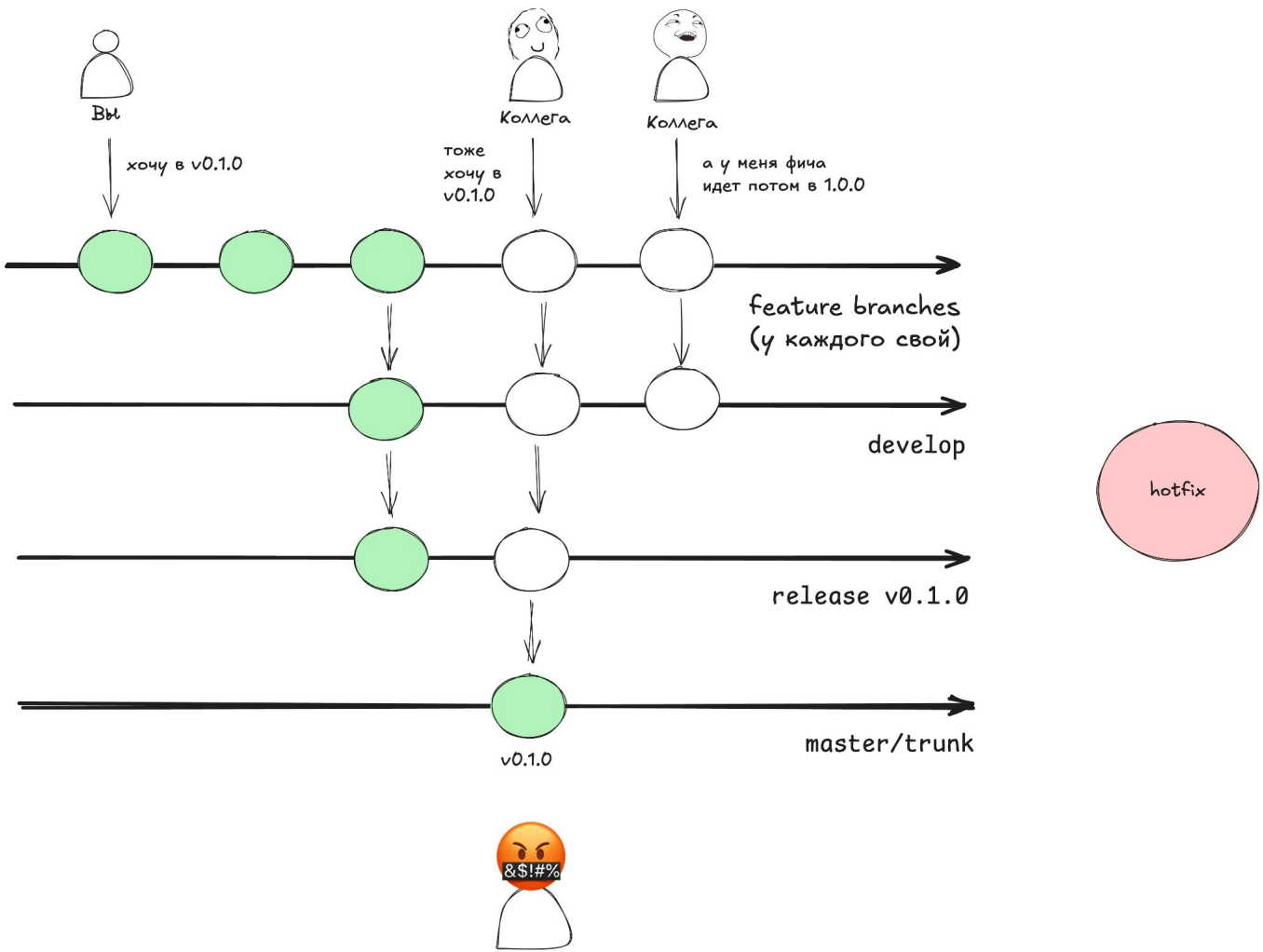

Вы
Коллега
Коллега
хочу в v0.1.0
тоже
хочу в
v0.1.0
а у меня фича
идет потом в 1.0.0
feature branches
(у каждого свой)
develop

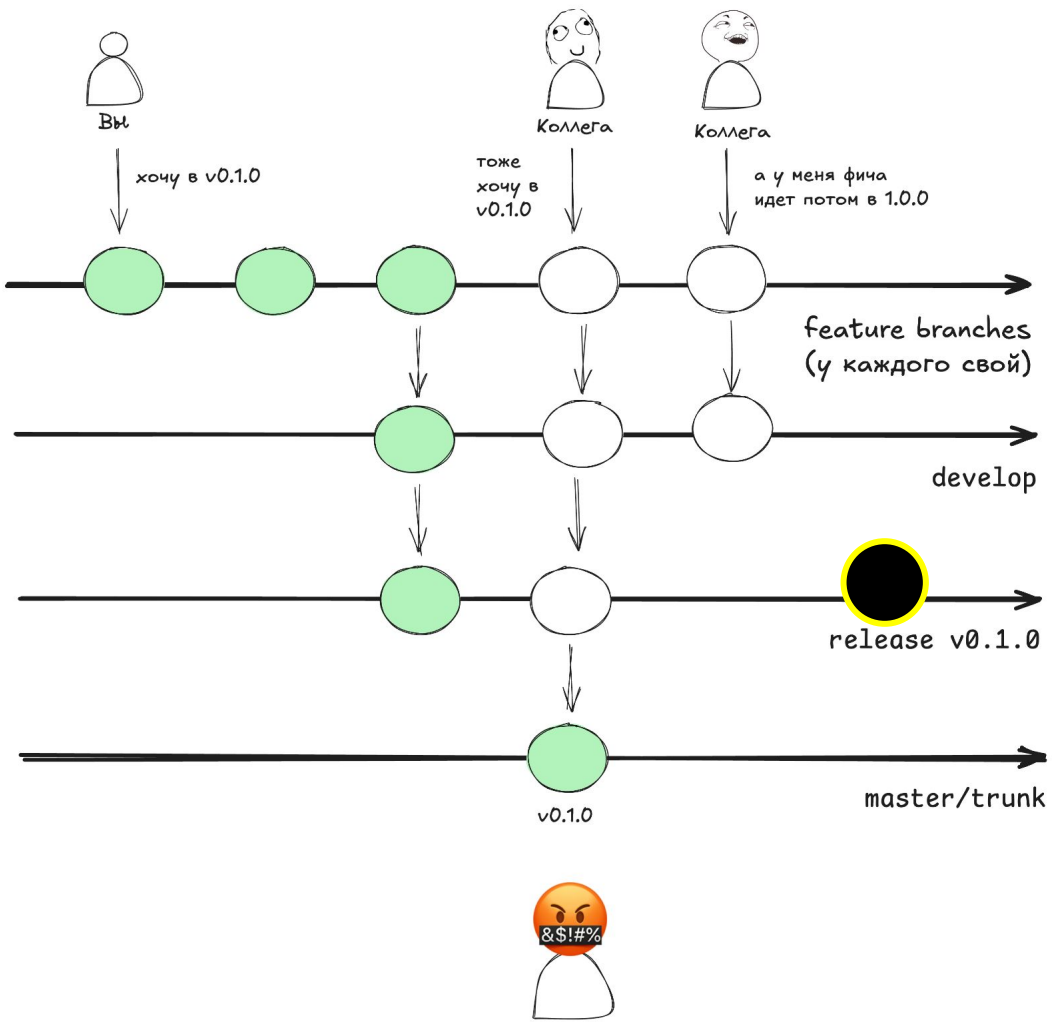

Вы
Коллега
Коллега
хочу в v0.1.0
тоже
хочу в
v0.1.0
а у меня фича
идет потом в 1.0.0
feature branches
(у каждого свой)
develop
release v0.1.0

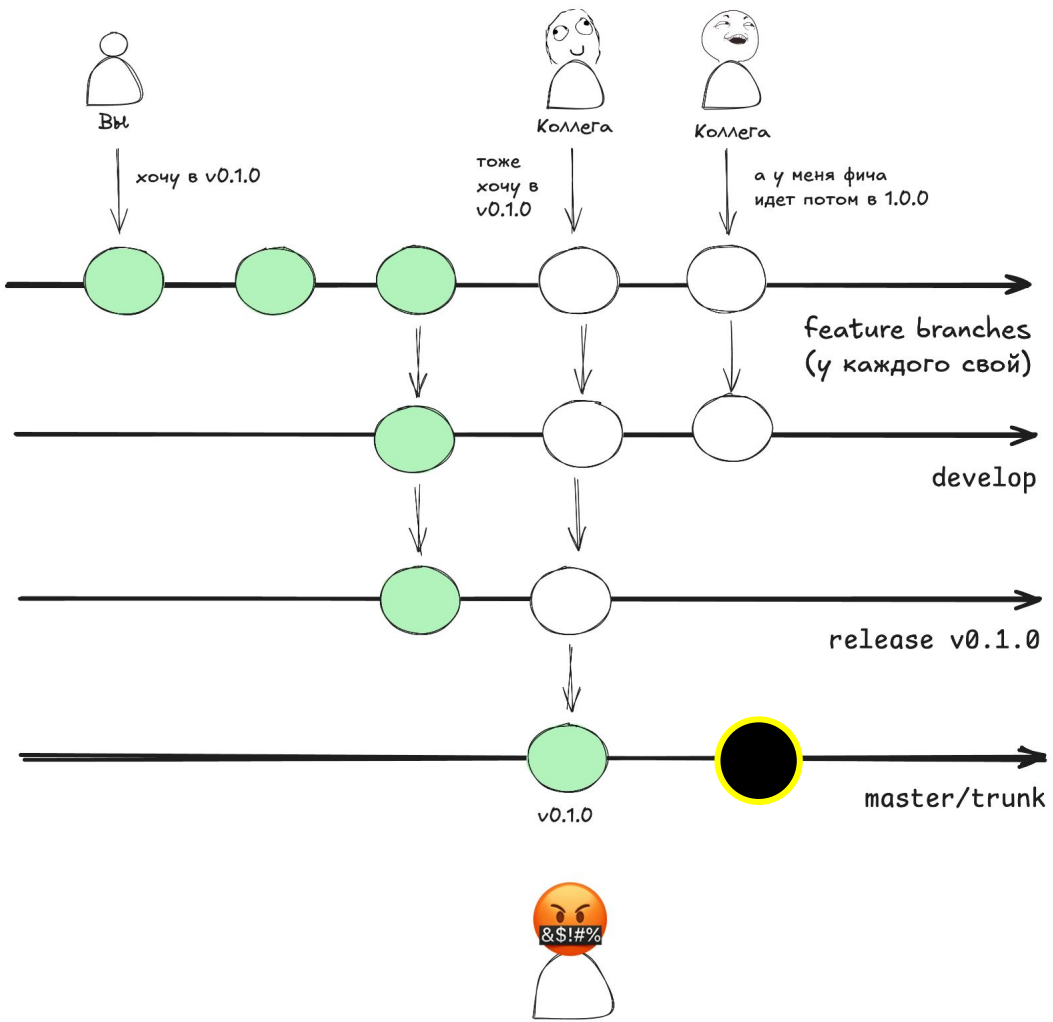

Вы
Коллега
Коллега
хочу в v0.1.0
тоже
хочу в
v0.1.0
а у меня фича
идет потом в 1.0.0
feature branches
(у каждого свой)
develop
release v0.1.0
master/trunk
v0.1.0

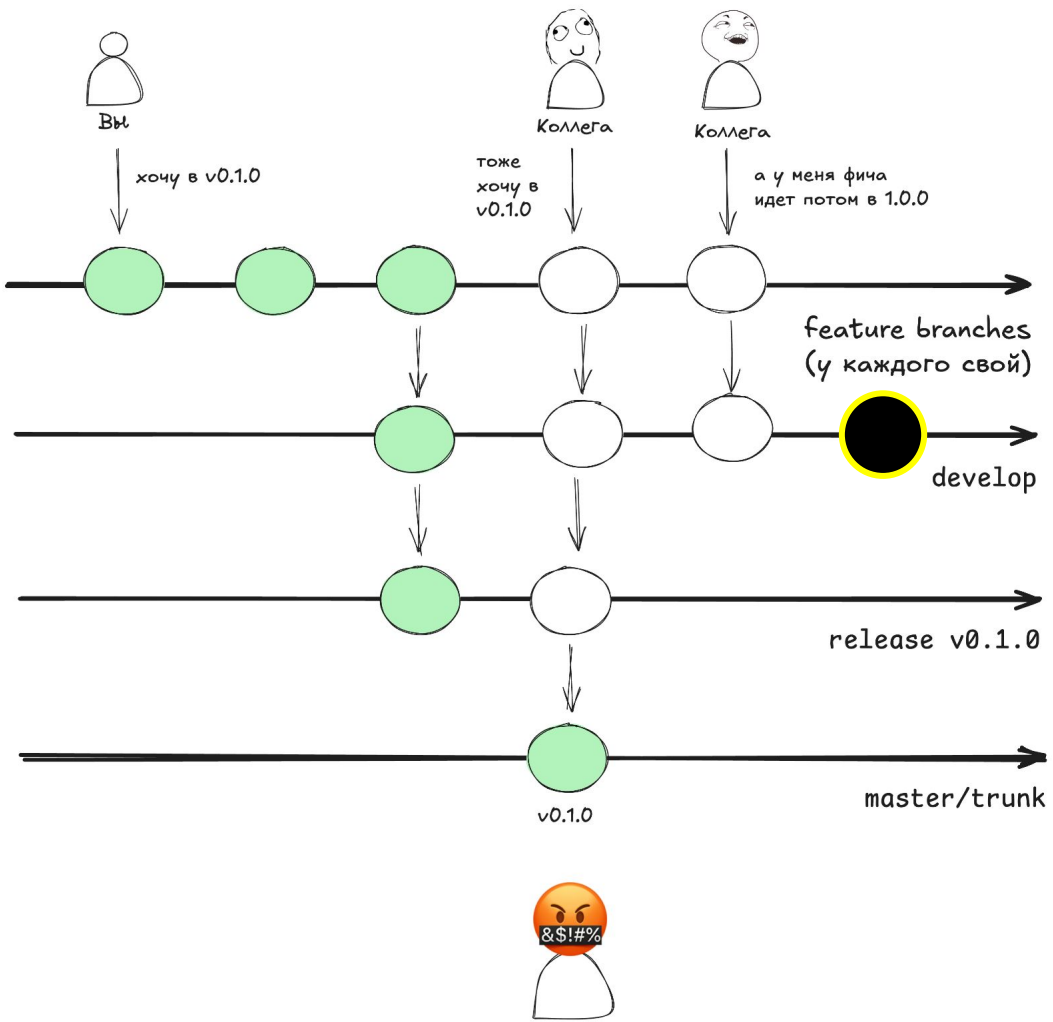

Вы
Коллега
Коллега
хочу в v0.1.0
тоже хочу в v0.1.0
а у меня фича идет потом в 1.0.0
feature branches (у каждого свой)
develop
release v0.1.0
master/trunk
v0.1.0

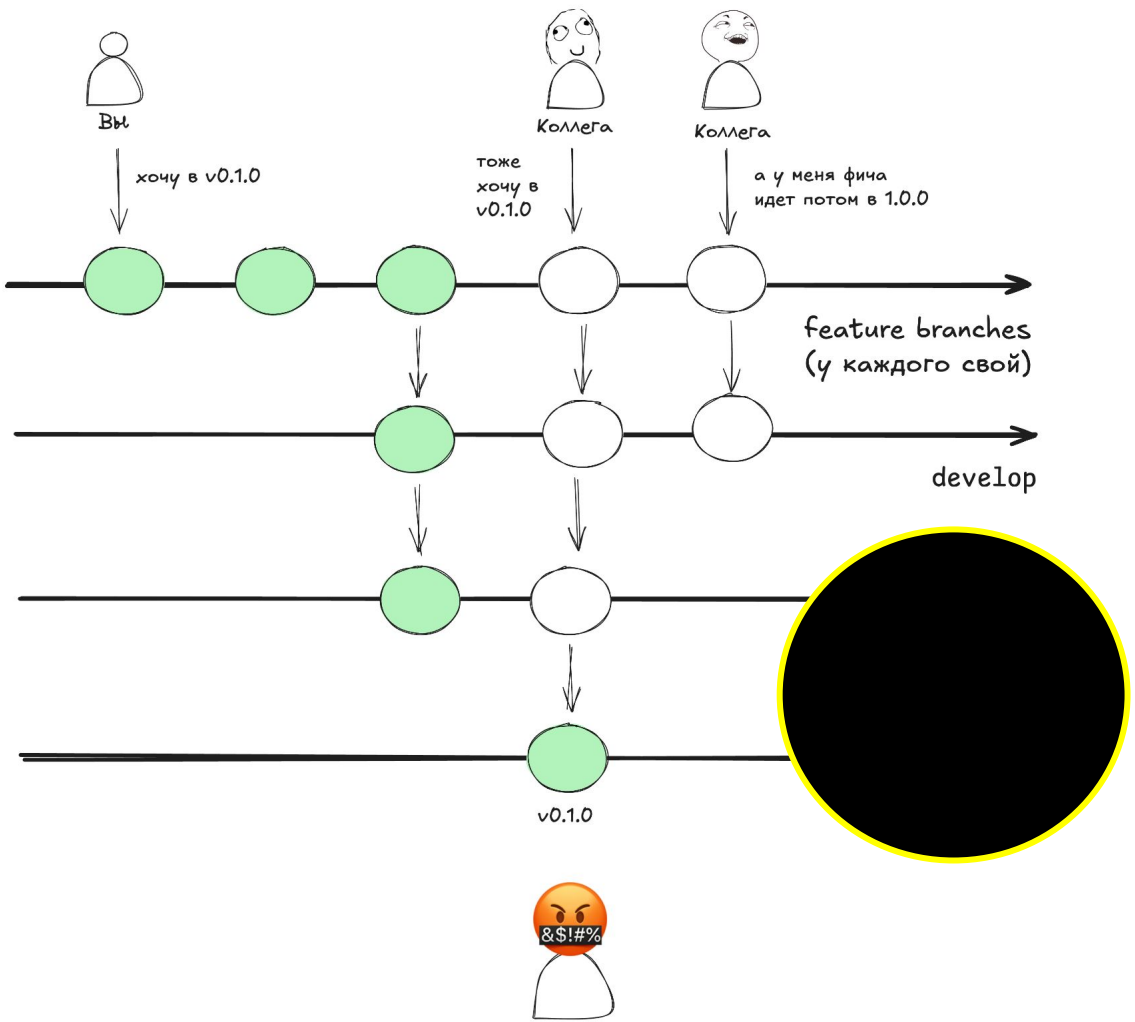
&$!#%

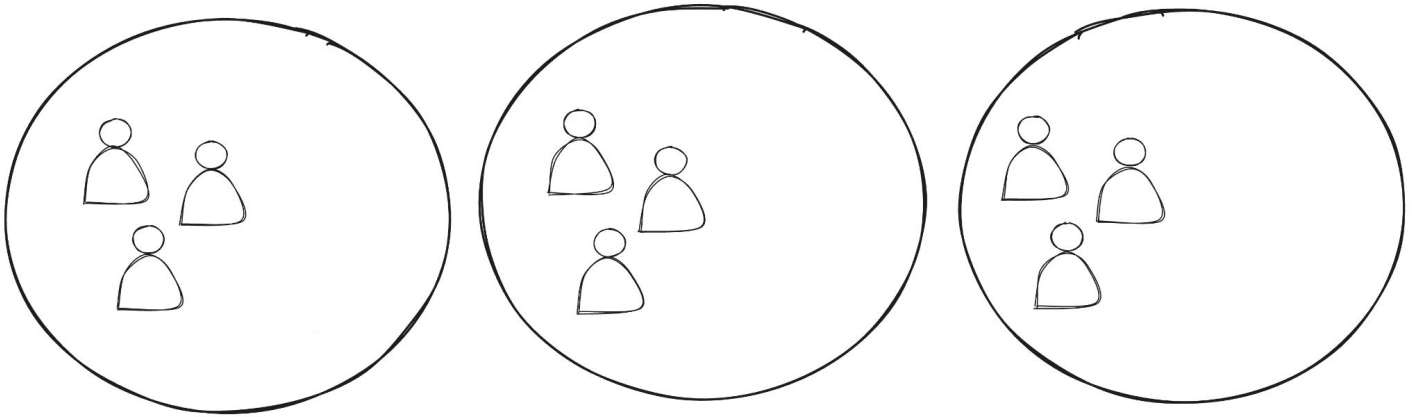

Вы
Коллега
Коллега
хочу в v0.1.0
тоже хочу в v0.1.0
а у меня фича идет потом в 1.0.0
feature branches (у каждого свой)
develop
hotfix
release v0.1.0
v0.1.0
master/trunk
&$!#%

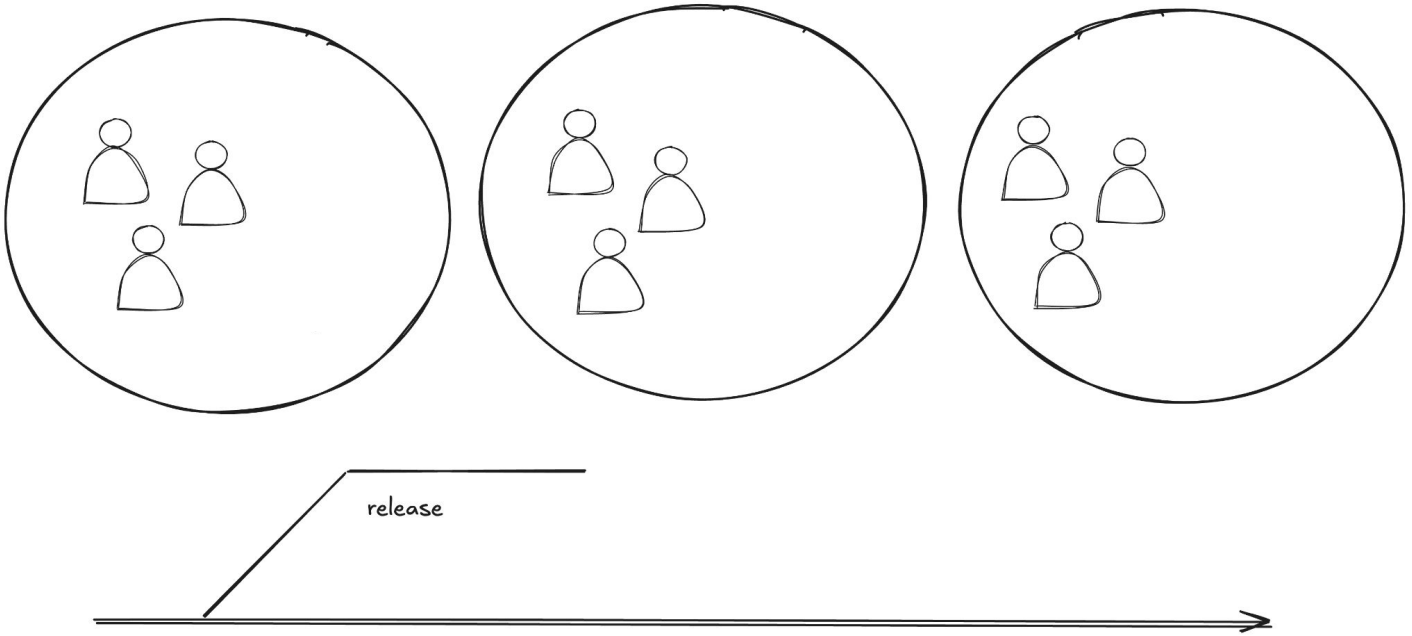

Вы
Коллега
Коллега
хочу в v0.1.0
тоже хочу в v0.1.0
а у меня фича идет потом в 1.0.0
feature branches (у каждого свой)
develop
release v0.1.0
master/trunk
v0.1.0
&$!#%

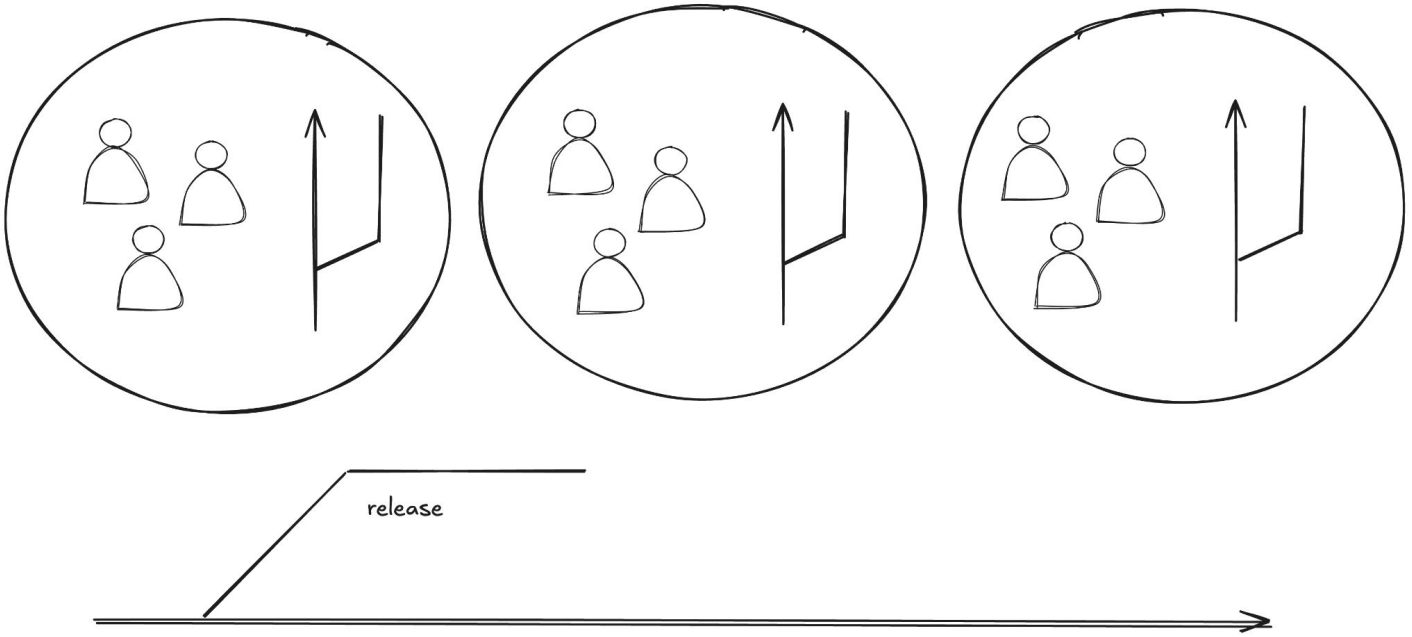

Вы
Коллега
Коллега
хочу в v0.1.0
тоже хочу в v0.1.0
а у меня фича идет потом в 1.0.0
feature branches (у каждого свой)
develop
release v0.1.0
master/trunk
v0.1.0
&$!#%

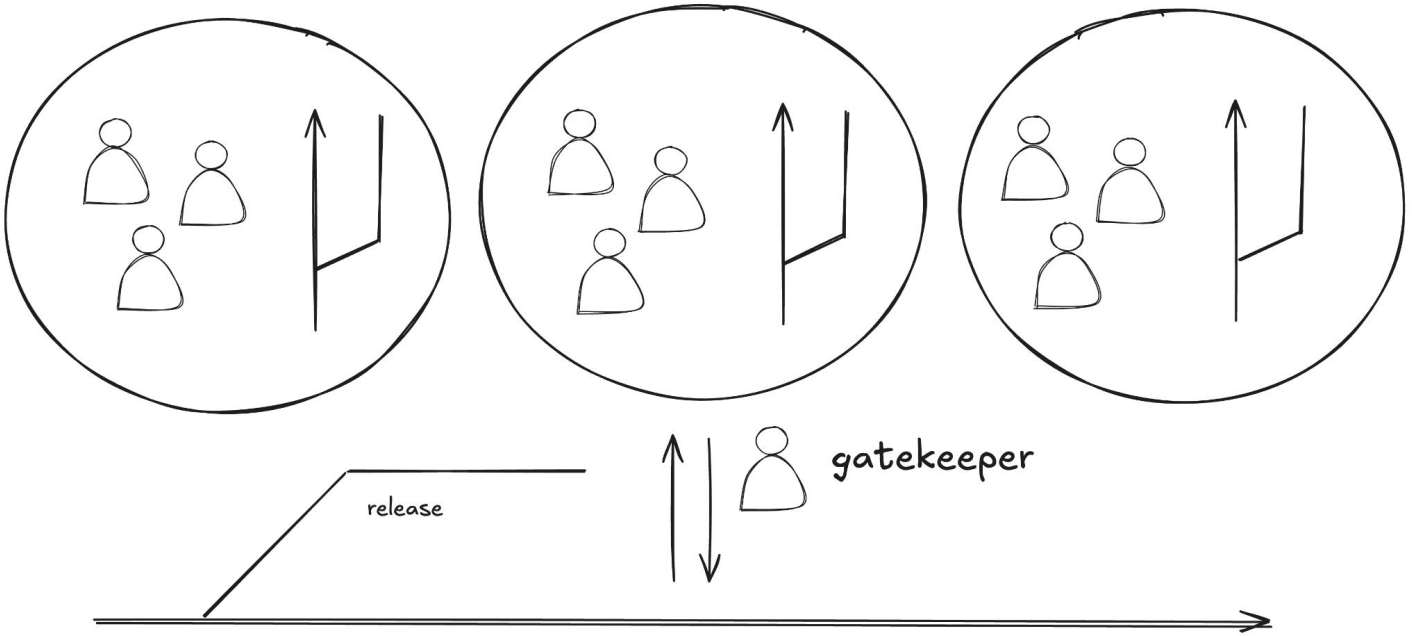

Вы
Коллега
Коллега
хочу в v0.1.0
тоже хочу в v0.1.0
а у меня фича идет потом в 1.0.0
feature branches (у каждого свой)
develop
release v0.1.0
master/trunk
v0.1.0
&$!#%


Вы
Коллега
Коллега
хочу в v0.1.0
тоже хочу в v0.1.0
а у меня фича идет потом в 1.0.0
feature branches (у каждого свой)
develop
v0.1.0
&$!#%




release


release


gatekeeper
release

A successful Git branching model

By Vincent Driessen
on Tuesday, January 05, 2010

Note of reflection (March 5, 2020)

This model was conceived in 2010, now more than 10 years ago, and not very long after Git itself came into being. In those 10 years, git-flow (the branching model laid out in this article) has become hugely popular in many a software team to the point where people have started treating it like a standard of sorts — but unfortunately also as a dogma or panacea.

During those 10 years, Git itself has taken the world by a storm, and the most popular type of software that is being developed with Git is shifting more towards web apps — at least in my filter bubble. Web apps are typically continuously delivered, not rolled back, and you don't have to support multiple versions of the software running in the wild.

This is not the class of software that I had in mind when I wrote the blog post 10 years ago. If your team is doing continuous delivery of software, I would suggest to adopt a much simpler workflow (like GitHub flow) instead of trying to shoehorn git-flow into your team.

If, however, you are building software that is explicitly versioned, or if you need to support multiple versions of your software in the wild, then git-flow may still be as good of a fit to your team as it has been to people in the last 10 years. In that case, please read on.

To conclude, always remember that panaceas don't exist. Consider your own context. Don't be hating. Decide for yourself.

https://nvie.com/posts/a-successful-git-branching-model/

Git Flow

- Монументальный сложный легаси процесс

Git Flow

- Монументальный сложный легаси процесс
- Merge-конфликты, pull-up, pull-down

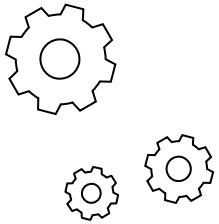
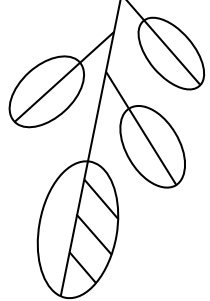
Git Flow

- Монументальный сложный легаси процесс
- Merge-конфликты, pull-up, pull-down
- Для Java-стека нет нормального тулинга CI/CD

Git Flow

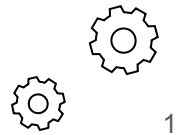
- Монументальный сложный легаси процесс
- Merge-конфликты, pull-up, pull-down
- Для Java-стека нет нормального тулинга CI/CD
- Тем, кто возит приложение клиенту на флешке

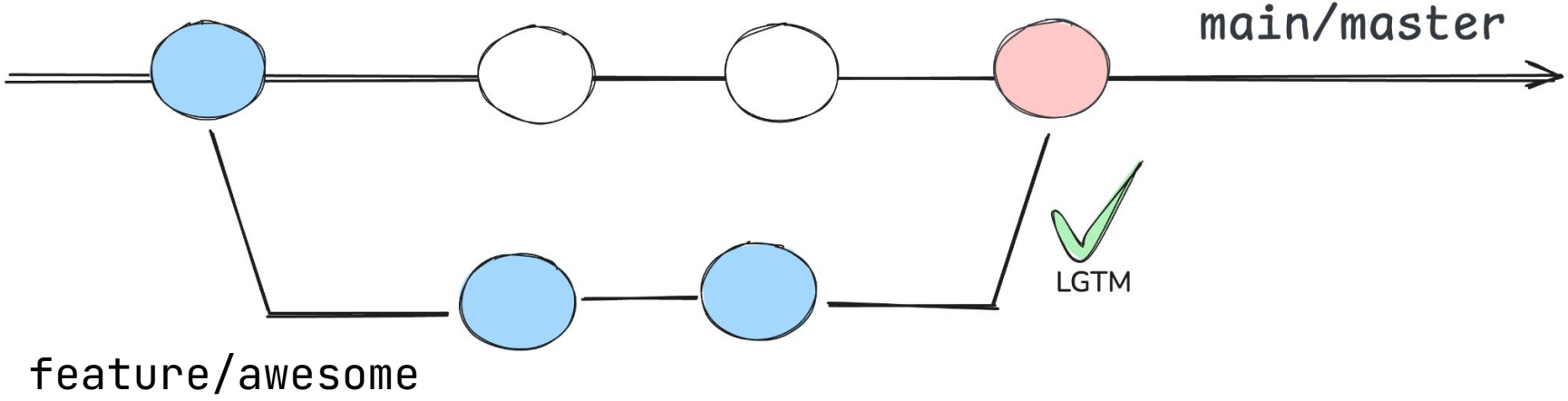
GitHub Flow

GitHub Flow

Feature branching


main/master
LGTM
feature/awesome

GitHub Flow

- Очень простой flow для начинающих

GitHub Flow

- Очень простой flow для начинающих
- Для Java есть весь тулинг, *чистый кайф с Reckon*

GitHub Flow

- Очень простой flow для начинающих
- Для Java есть весь тулинг, *чистый кайф с Reckon*
- Очень хорошо подходящий flow для web-apps, SaaS

GitHub Flow

- Очень простой flow для начинающих
- Для Java есть весь тулинг, *чистый кайф с Reckon*
- Очень хорошо подходящий flow для web-apps, SaaS
- Релизы и тэги делаются из основной ветки

GitHub Flow

- Очень простой flow для начинающих
- Для Java есть весь тулинг, *чистый кайф с Reckon*
- Очень хорошо подходящий flow для web-apps, SaaS
- Релизы и тэги делаются из основной ветки
- Что, если релизы по времени или с задержкой?

GitHub Flow

- Очень простой flow для начинающих
- Для Java есть весь тулинг, *чистый кайф с Reckon*
- Очень хорошо подходящий flow для web-apps, SaaS
- Релизы и тэги делаются из основной ветки
- Что, если релизы по времени или с задержкой?
- Иногда симбиоз с **Forking** Workflow

r/ExperiencedDevs • 1 yr. ago
rjm101

CTO is pushing for trunk based development, team is heavily against the idea, what to do?

So we have a fairly new CTO thats pushing for various different process changes in dev teams.

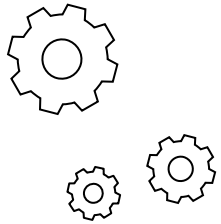
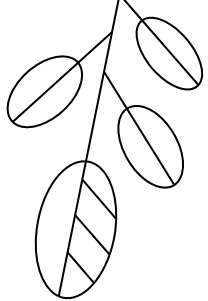
Two of these is trunk based development and full time pair programming to enable CI/CD.

For context my team looks after a critical area of our platforms (the type where if we screw up serious money can be lost and we'll have regulators to answer to). We commit to repos that are contributed to by multiple teams and basically use a simplified version of Gitflow with feature branches merging into master only when fully reviewed & tested and considered prod ready. Once merged to master the change is released to prod.

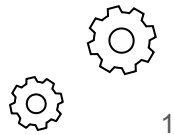
From time to time we do pair programming but tend to only do it when it's crunch time where necessary. The new process basically wants this full time. Devs have trialed this and feel burned out doing the pair programming all day everyday.

Basically I ran my team on the idea of trunk based development and they're heavily against it including the senior devs (one of whom called it 'madness').

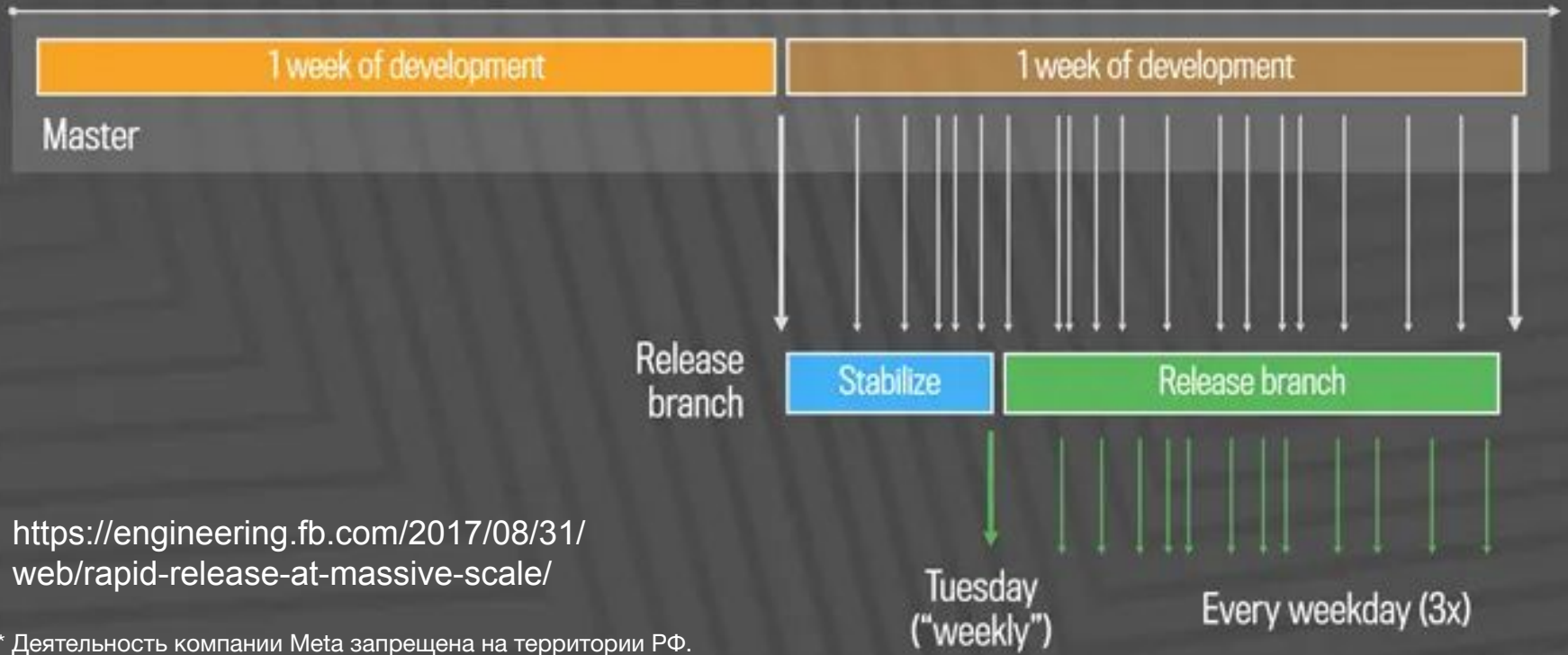
The main issue from their perspective is they consider it risky and few others don't think it will actually improve anything. I'm not entirely clued up on where manual QA testing fits into the process either but what I've read suggests this takes place after merge to master & even release which is a big concern for the team. Devs know that manual QA's capture important bugs via non-happy paths despite having a lot of automated tests and 100% code coverage. We already use feature flags for our projects so that we only expose this to clients when ready but devs know this isn't full proof.

Trunk Based Frontend

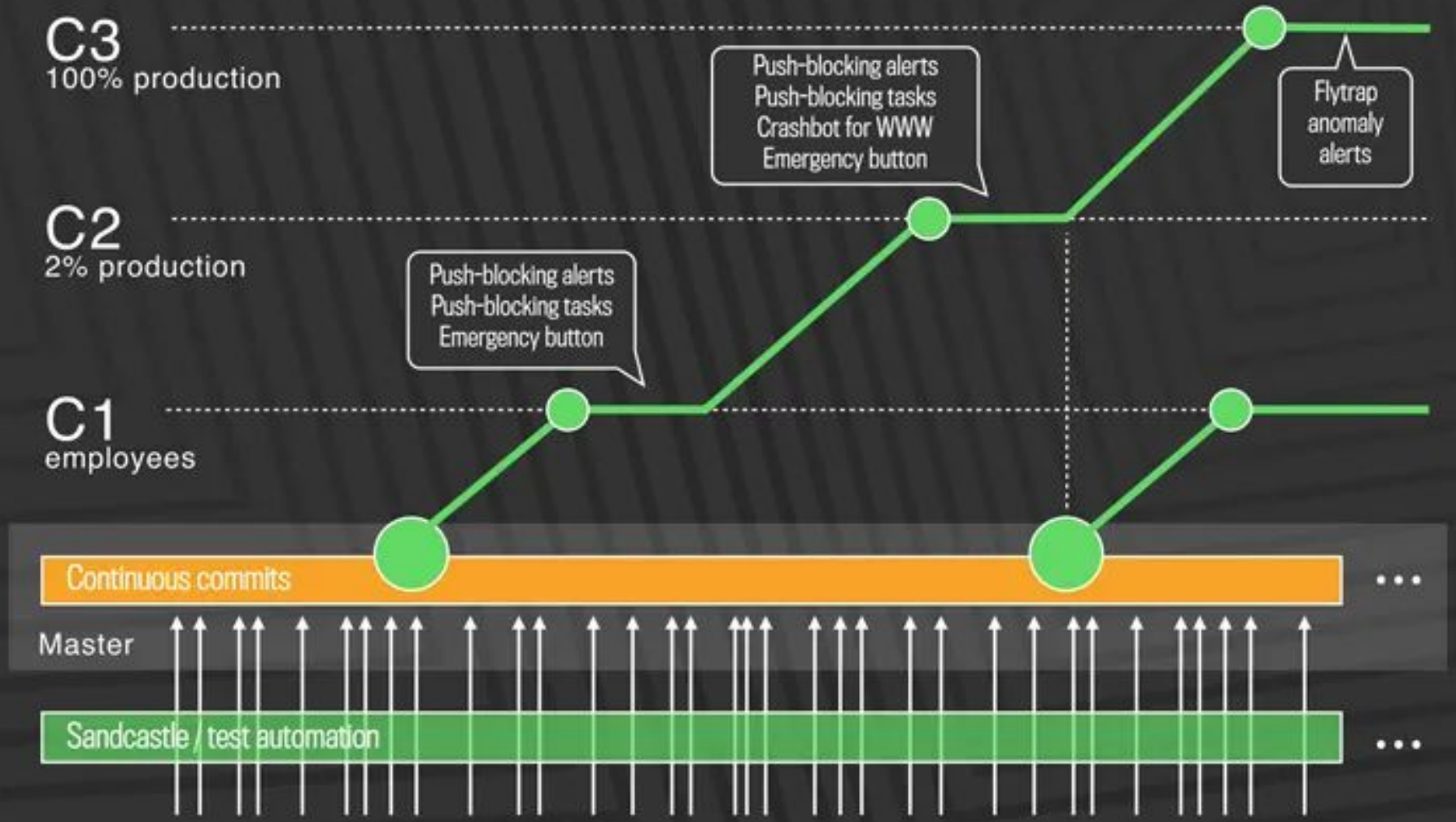



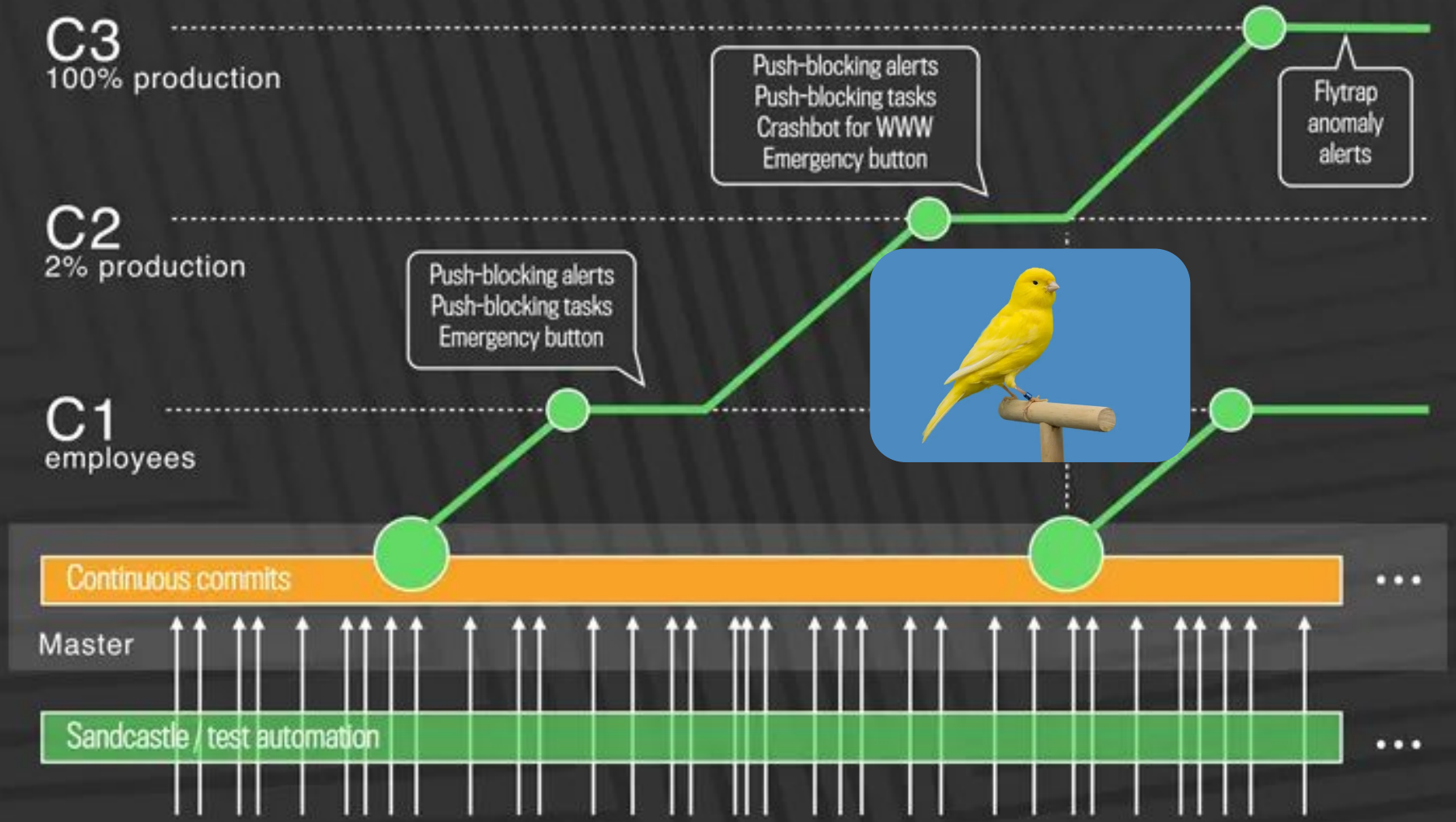
www.facebook.com

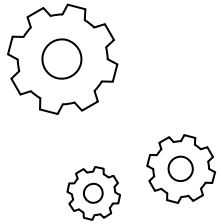
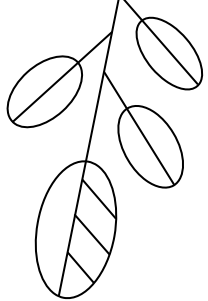


https://engineering.fb.com/2017/08/31/web/rapid-release-at-massive-scale/

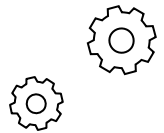
* Деятельность компании Meta запрещена на территории РФ.


C3
100% production
Push-blocking alerts
Push-blocking tasks
Crashbot for WWW
Emergency button
Flytrap
anomaly
alerts
C2
2% production
Push-blocking alerts
Push-blocking tasks
Emergency button
C1
employees
Continuous commits
Master
Sandcastle / test automation

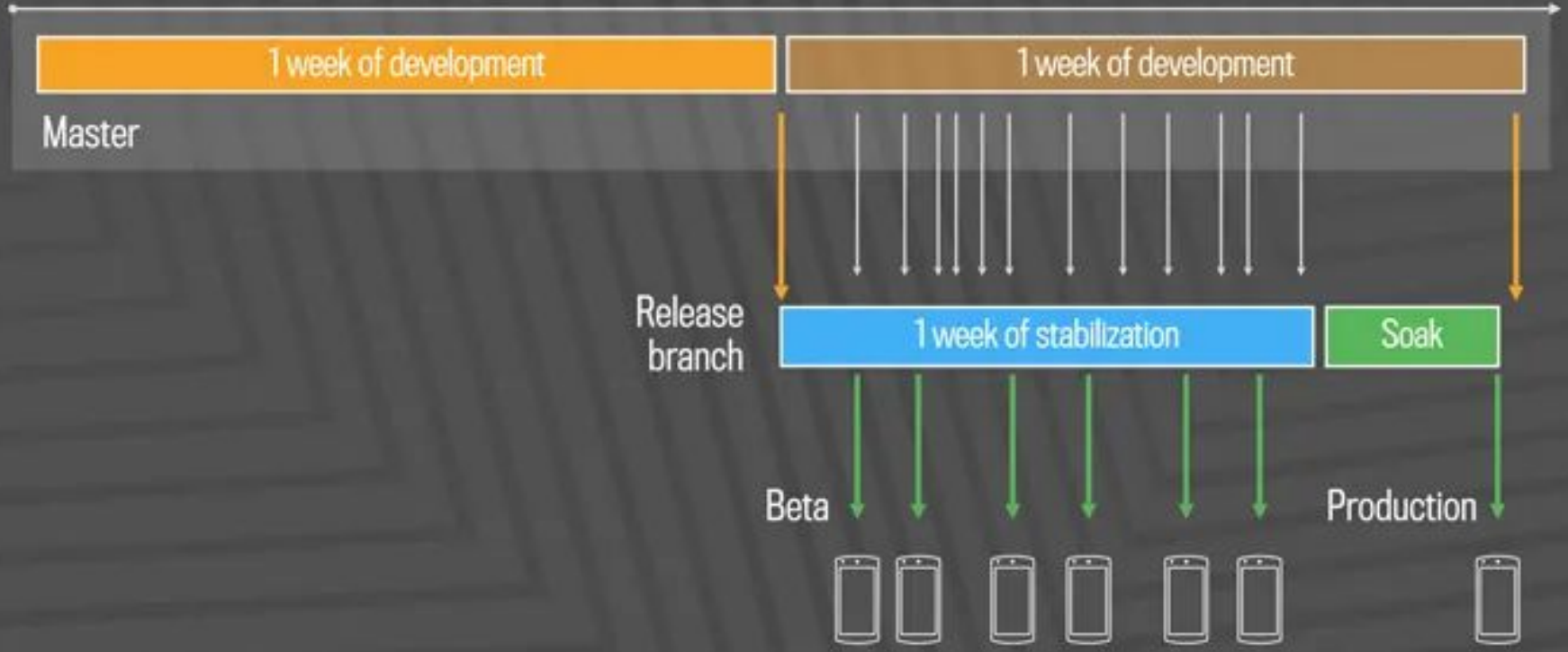

C3
100% production
Push-blocking alerts
Push-blocking tasks
Crashbot for WWW
Emergency button
Flytrap
anomaly
alerts
C2
2% production
Push-blocking alerts
Push-blocking tasks
Emergency button
C1
employees
Continuous commits
Master
Sandcastle / test automation

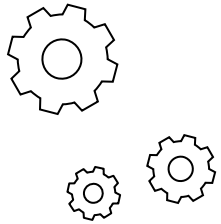
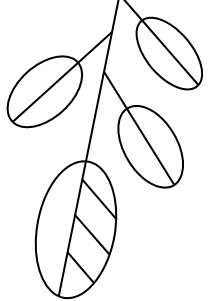



Trunk Based Mobile App

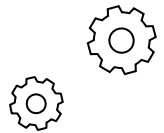



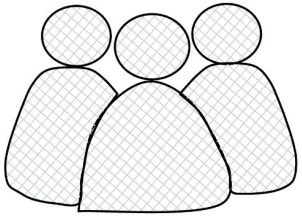
Native mobile



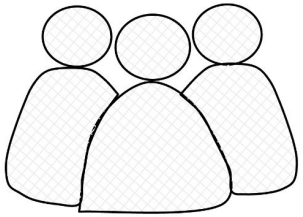



Trunk Based для Java BE

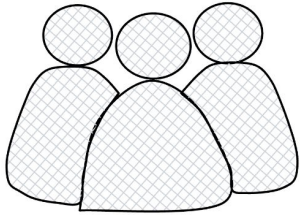





Clients



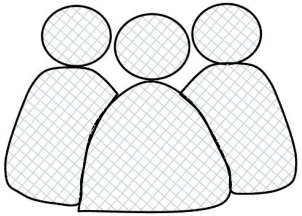
Clients



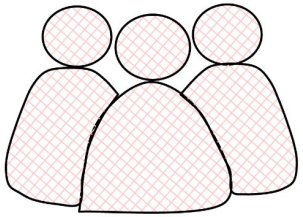
Clients

Java
Backend
V 1

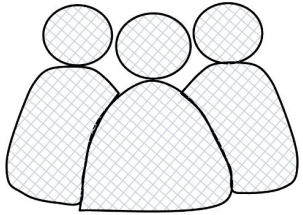
Java
Backend
V 1



Clients



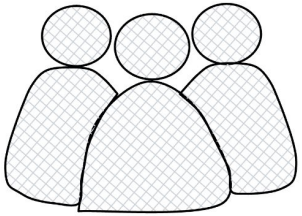
Clients



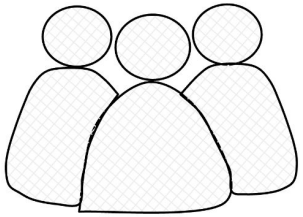
Clients

Java
Backend
V 1

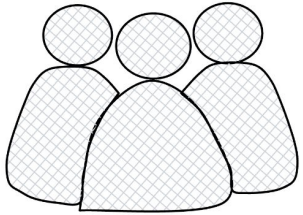
Java
Backend
V 1



Clients



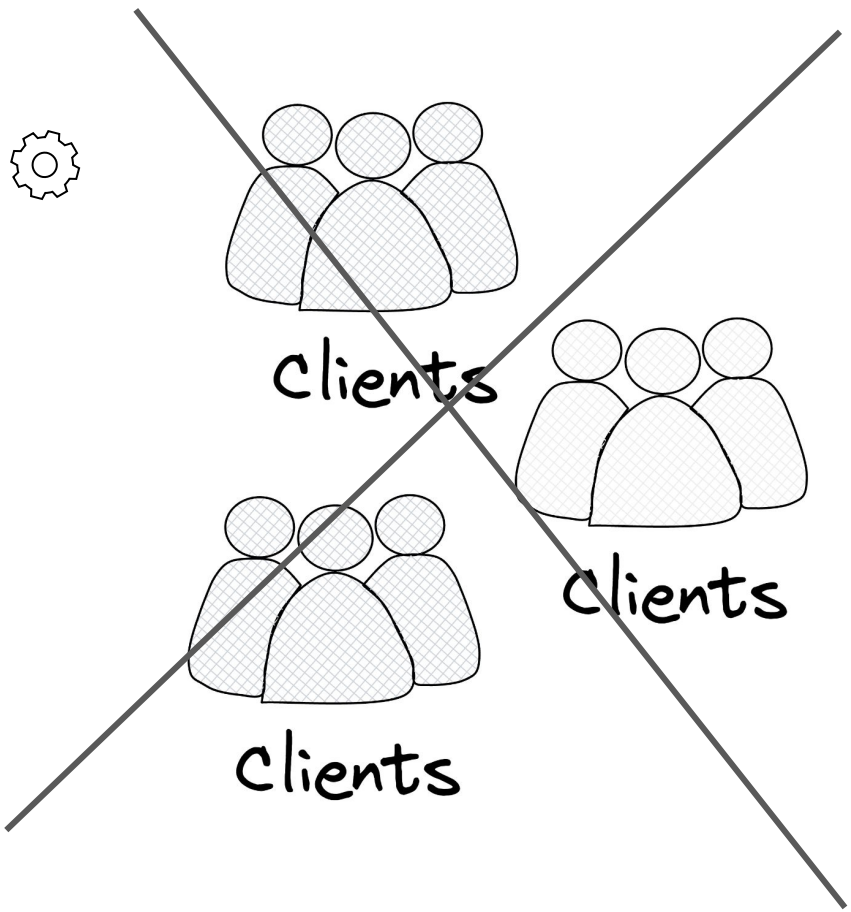
Clients



Clients

Java Backend V 1

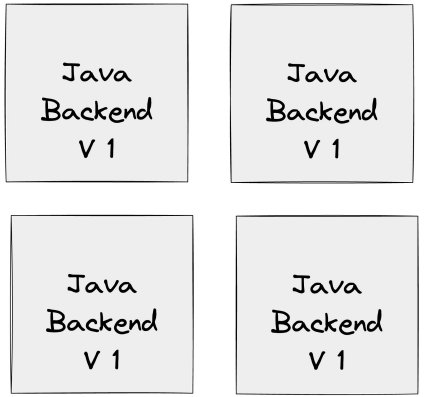
Java Backend V 1



Java
Backend
V 1

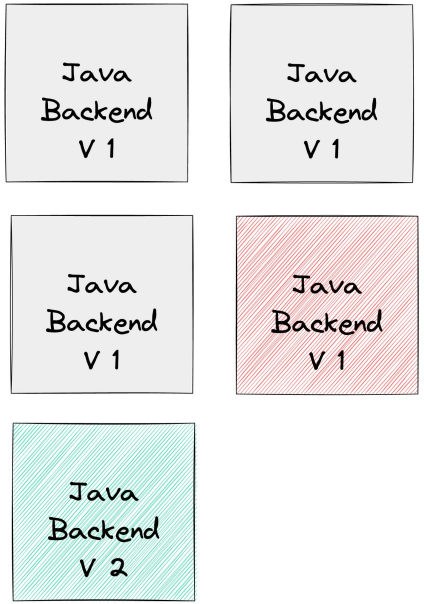
Java
Backend
V 1

В современном k8s-мире

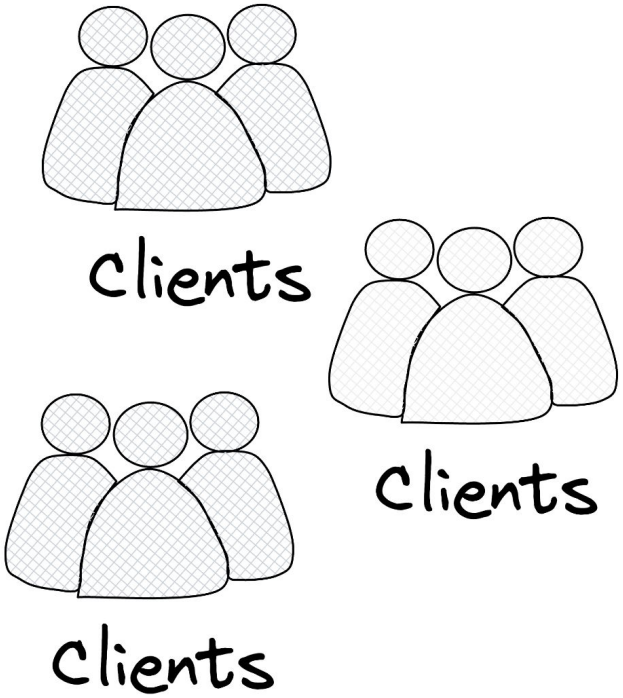


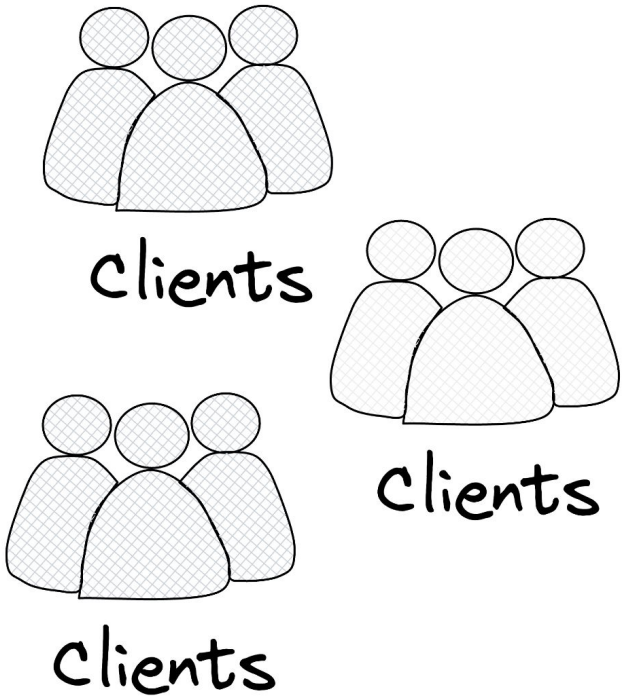

```yaml
spec:
  strategy:
    type: RollingUpdate
  rollingUpdate:
    maxSurge: 25%
    maxUnavailable: 25%
```

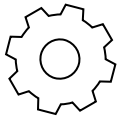
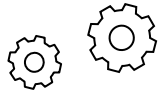
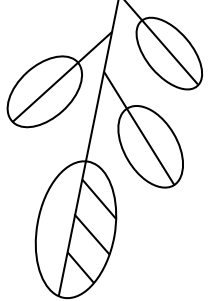
В современном k8s-мире




```yaml
spec:
  strategy:
    type: RollingUpdate
  rollingUpdate:
    maxSurge: 25%
    maxUnavailable: 25%
```

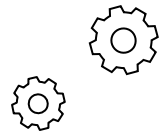

Clients
Clients
Clients
L4/L7
LB
Java
Backend
V 1
Java
Backend
V 1


Clients
Clients
Clients
L4/L7
LB
Java
Backend
V 1
Java
Backend
V 2

TBD Flow

- Сколько у вас разработчиков в проекте?

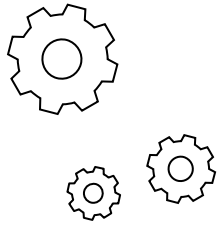
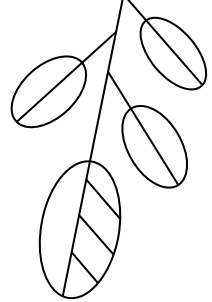
TBD Flow

- Сколько у вас разработчиков в проекте?

- Вы точно сможете отдельные фичи в вашем Spring приложении оборачивать в feature-flags?

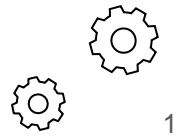
TBD Flow

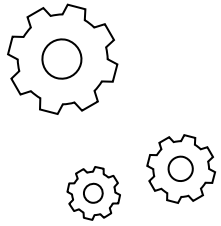
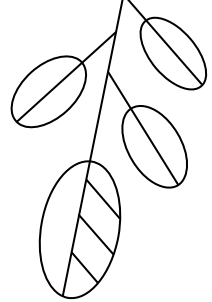
- Сколько у вас разработчиков в проекте?
- Вы точно сможете отдельные фичи в вашем Spring приложении оборачивать в feature-flags?
- Красивое версионирование неактуально: хоть commitId бери, хоть timestamp

TBD Flow

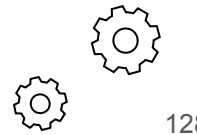
- Сколько у вас разработчиков в проекте?
- Вы точно сможете отдельные фичи в вашем Spring приложении оборачивать в feature-flags?
- Красивое версионирование неактуально: хоть commitId бери, хоть timestamp
- Точно ли подойдет для публичных вещей?

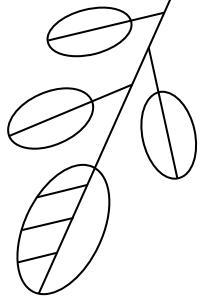



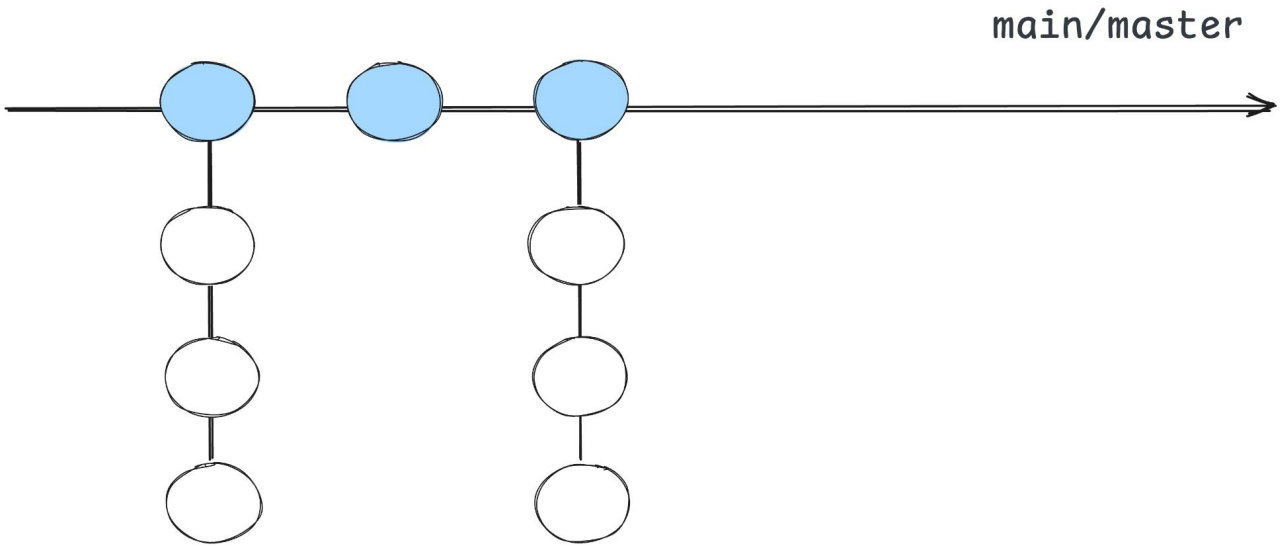
TBD Flow

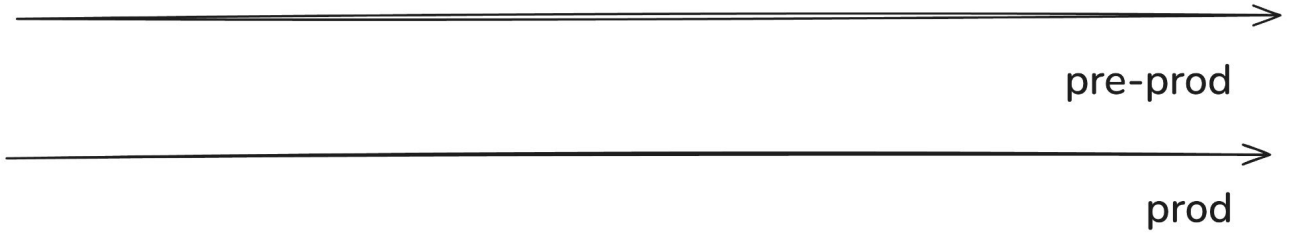
- Сколько у вас разработчиков в проекте?
- Вы точно сможете отдельные фичи в вашем Spring приложении оборачивать в feature-flags?
- Красивое версионирование неактуально: хоть commitId бери, хоть timestamp
- Точно ли подойдет для публичных вещей?
- Ресурсоемкая задача по организации Canary

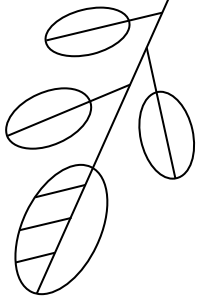



GitLab Flow




main/master
stable/release 1
stable/release 2

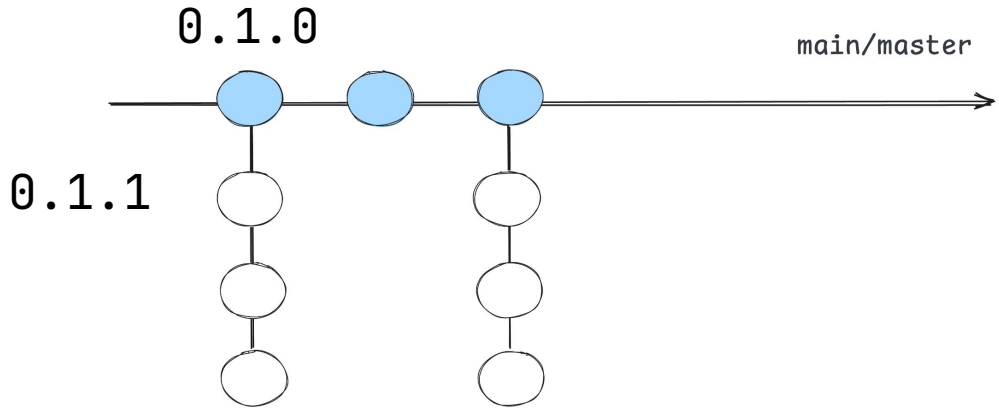

pre-prod
prod



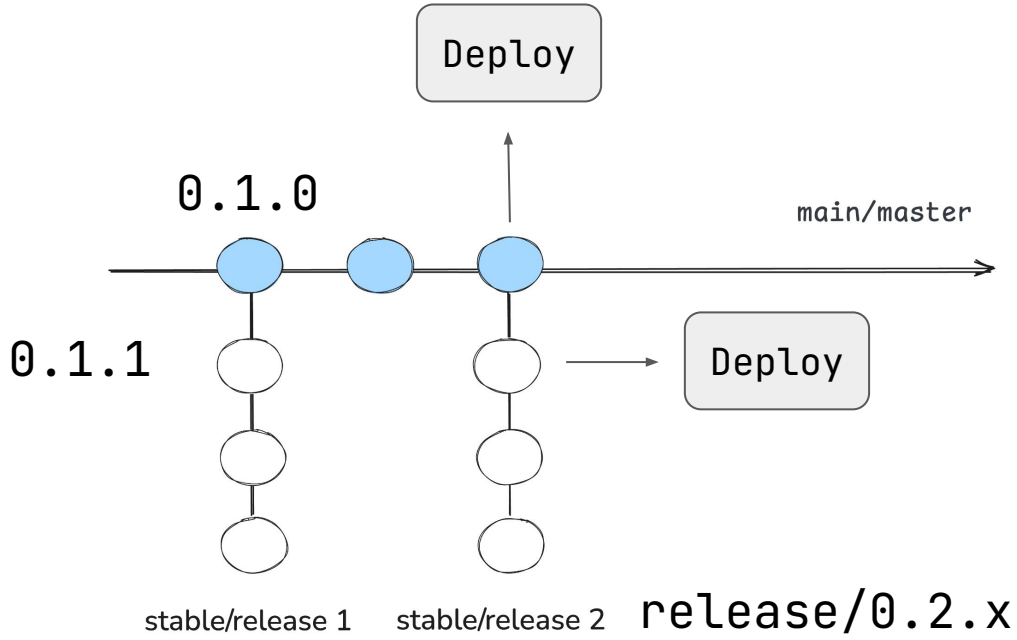
Советы от GitLab

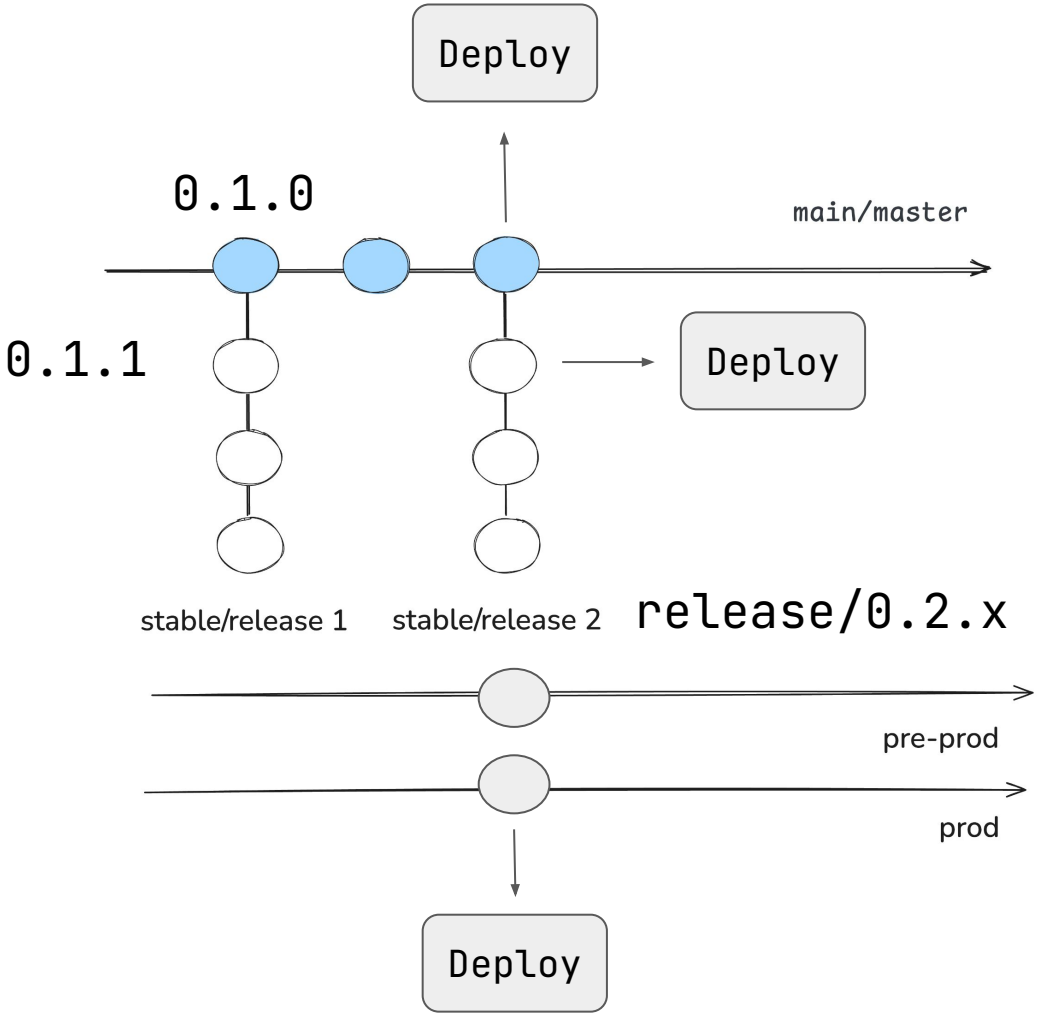
5. Deployments are automatic based on branches or tags.

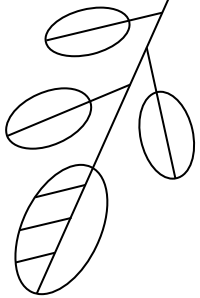
If developers don't want to deploy `main` every time, they can create a `production` branch. Rather than using a script or doing it manually, teams can use automation or have a specific branch that triggers a production deploy.



release/0.1.x stable/release 1 stable/release 2


Deploy
0.1.0
main/master
0.1.1
Deploy
stable/release 1
stable/release 2
release/0.2.x

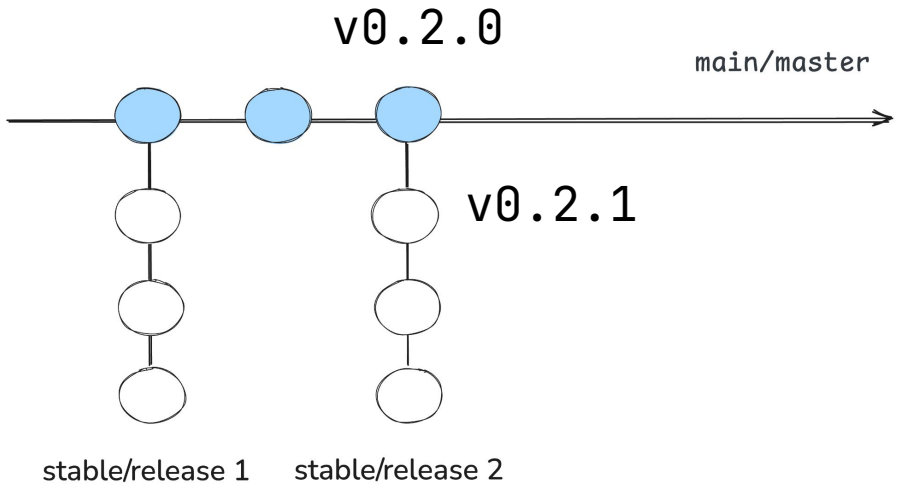

Deploy
0.1.0
main/master
0.1.1
Deploy
stable/release 1
stable/release 2
release/0.2.x
pre-prod
prod
Deploy

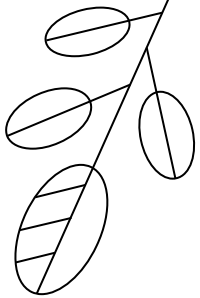


Советы от GitLab

6. **Tags** are set by the user, not by CI.

7. **Releases** are based on **tags**.

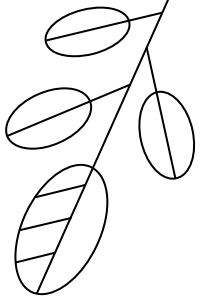

v0.2.0
main/master
v0.2.1
stable/release 1
stable/release 2



Советы от GitLab

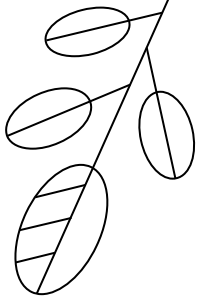
8. Pushed commits are **never rebased.**

10. Fix bugs in main first and release branches second.



GitLab Flow

- Atlassian советуют примерно также вести проект и делать релизы, но наоборот рекомендуют **rebase**

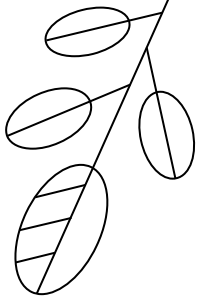


GitLab Flow

- Atlassian советуют примерно также вести проект и делать релизы, но наоборот рекомендуют **rebase**

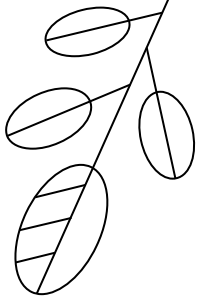
- GitLab Flow хорошо подходит для Java backend

- И особенно для **Enterprise** приложений

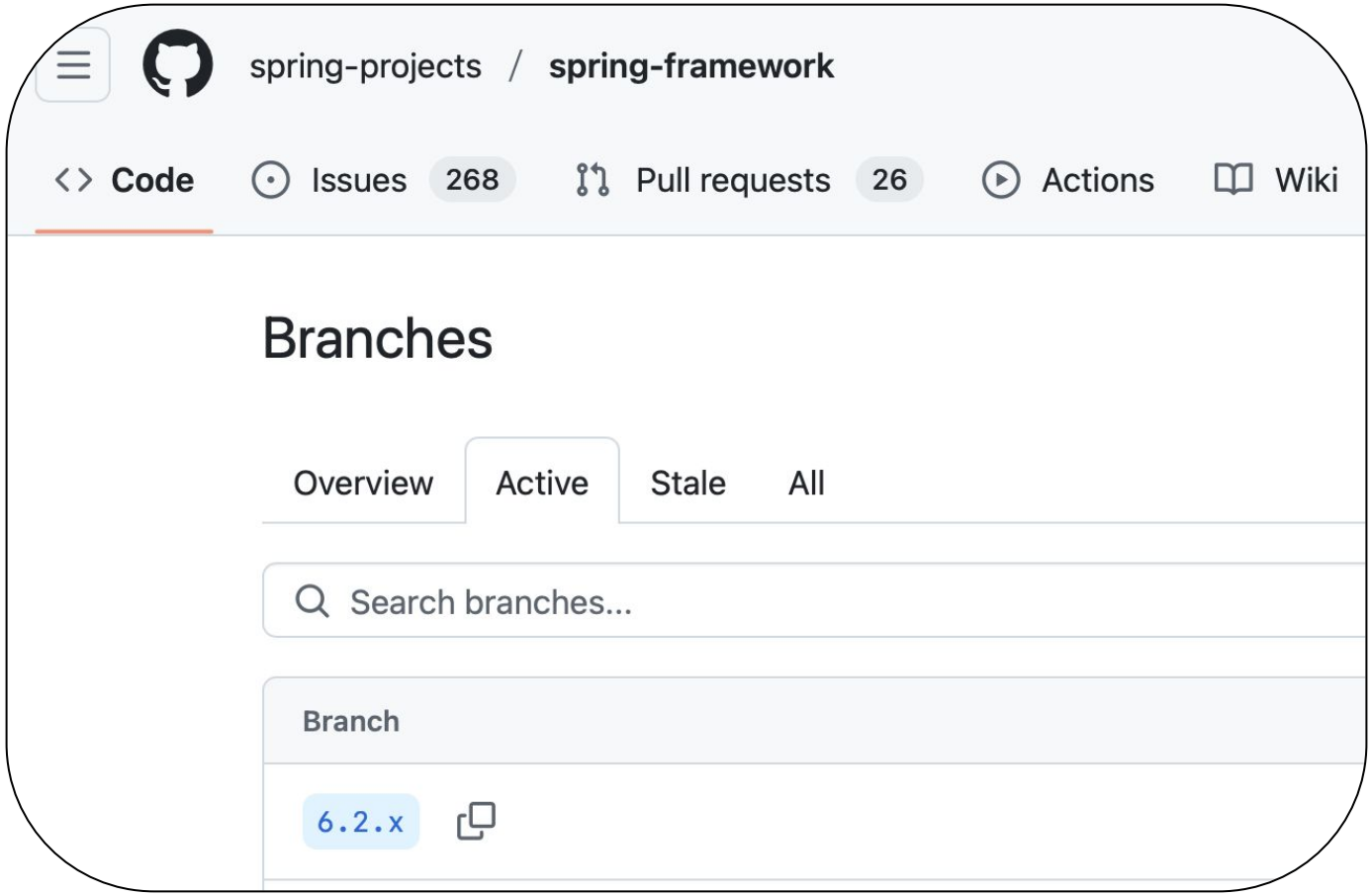


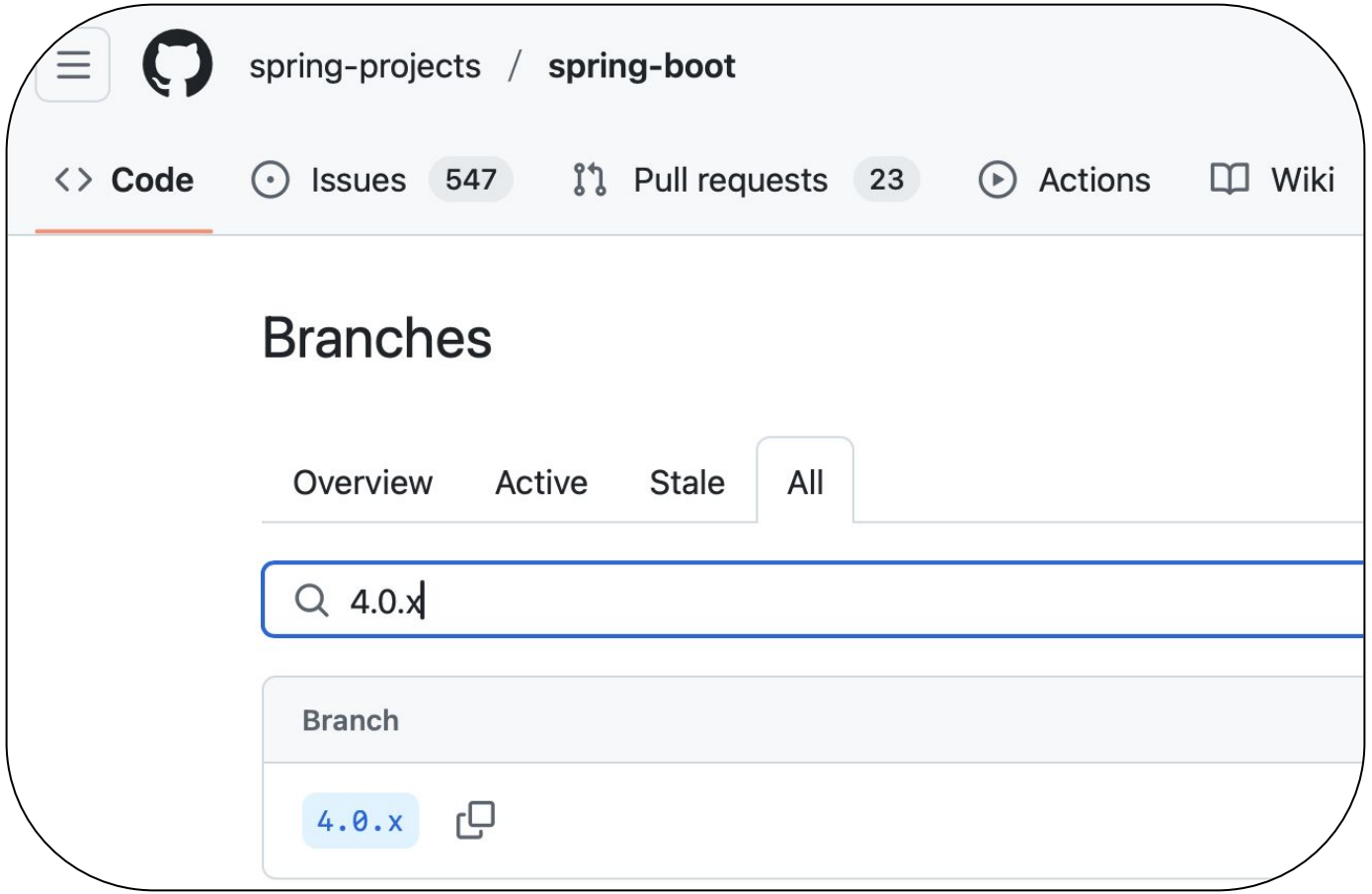
GitLab Flow

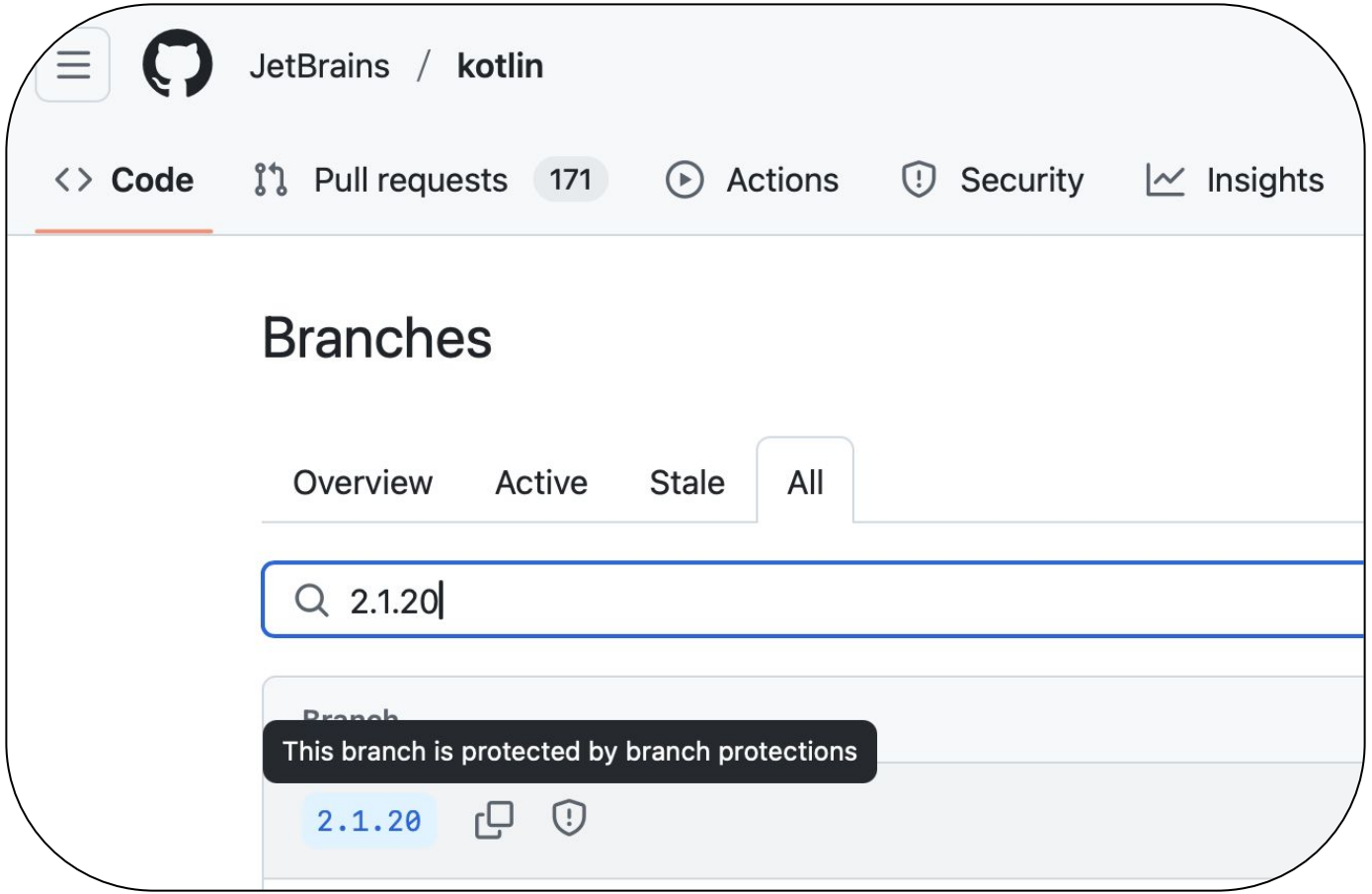
- Atlassian советуют примерно также вести проект и делать релизы, но наоборот рекомендуют **rebase**
- GitLab Flow хорошо подходит для Java backend
- И особенно для **Enterprise** приложений
- Автоматизации и версионирования мне не хватало или слишком сложные

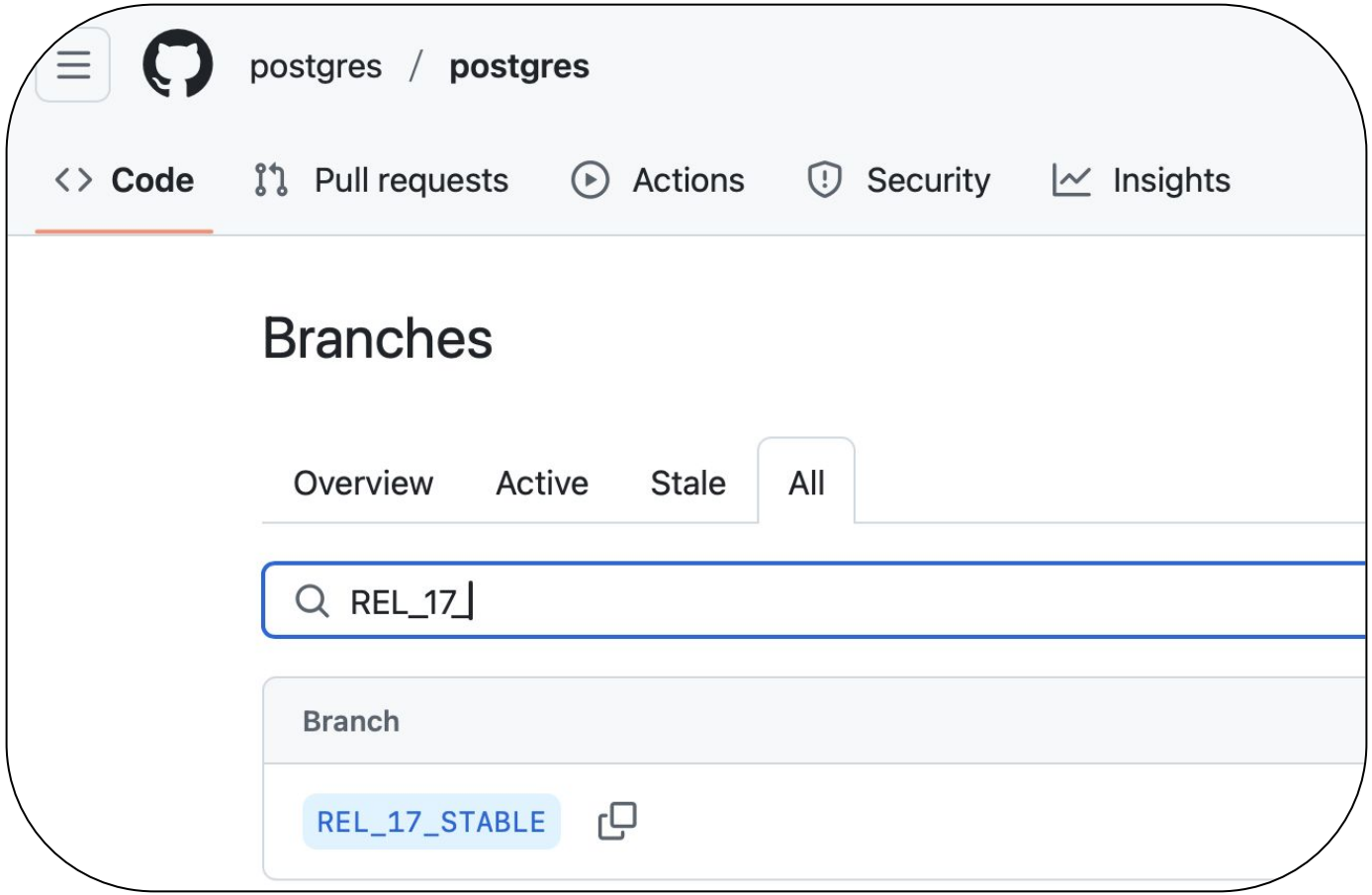


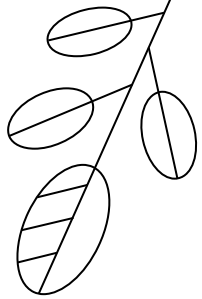
Как версионируются и выходят в релиз наши любимые проекты?


spring-projects / spring-framework
Code
Issues 268
Pull requests 26
Actions
Wiki
Branches
Overview
Active
Stale
All
Search branches...
Branch
6.2.x


spring-projects / spring-boot
Code
Issues 547
Pull requests 23
Actions
Wiki
Branches
Overview
Active
Stale
All
4.0.x
Branch
4.0.x


JetBrains / kotlin
Code
Pull requests 171
Actions
Security
Insights
Branches
Overview
Active
Stale
All
2.1.20
This branch is protected by branch protections
2.1.20

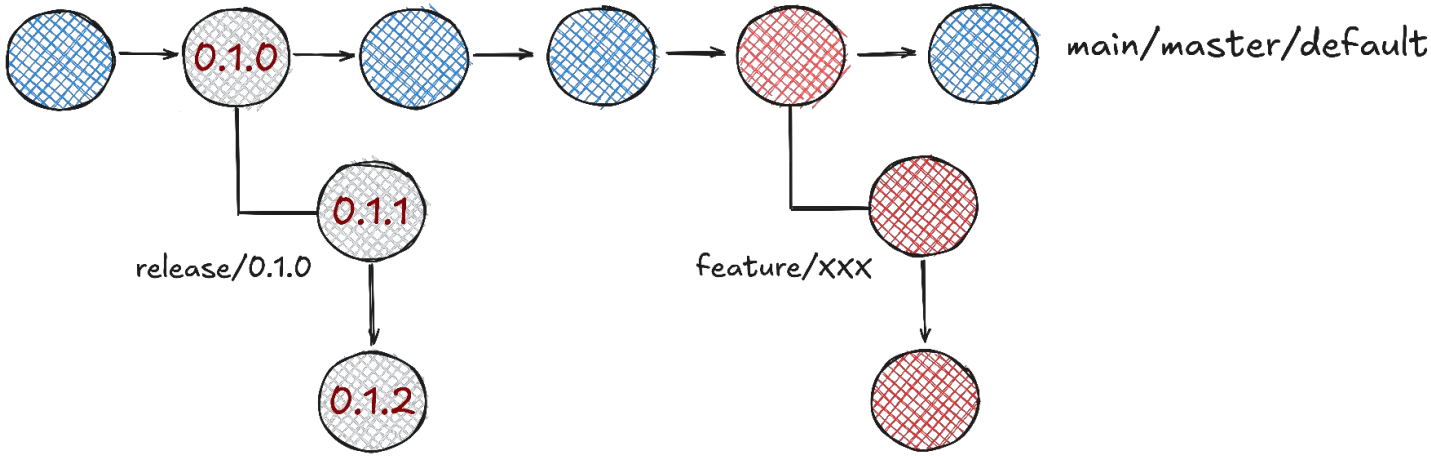

postgres / postgres
Code
Pull requests
Actions
Security
Insights
Branches
Overview
Active
Stale
All
REL_17_
Branch
REL_17_STABLE

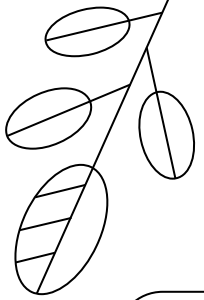


Что если строить процесс версионирования и релизов на комбинации **веток** и **тегов**?

https://github.com/**orchestr7**/vercraft

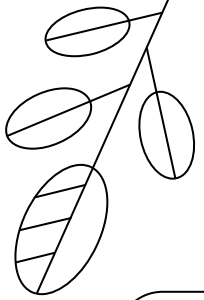
gitVersion task





VerCraft

```
plugins {
    id("com.akuleshov7.vercraft.plugin-gradle") version("0.5.0")
}
```

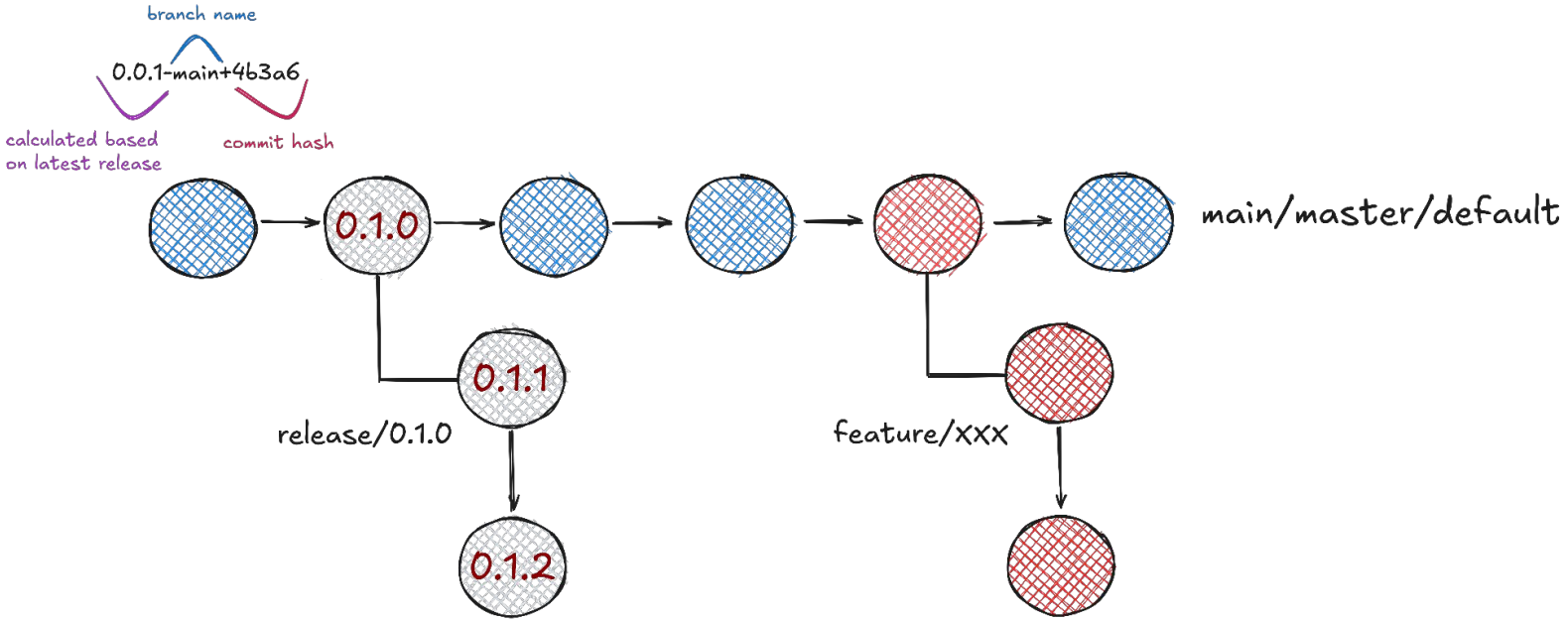


VerCraft

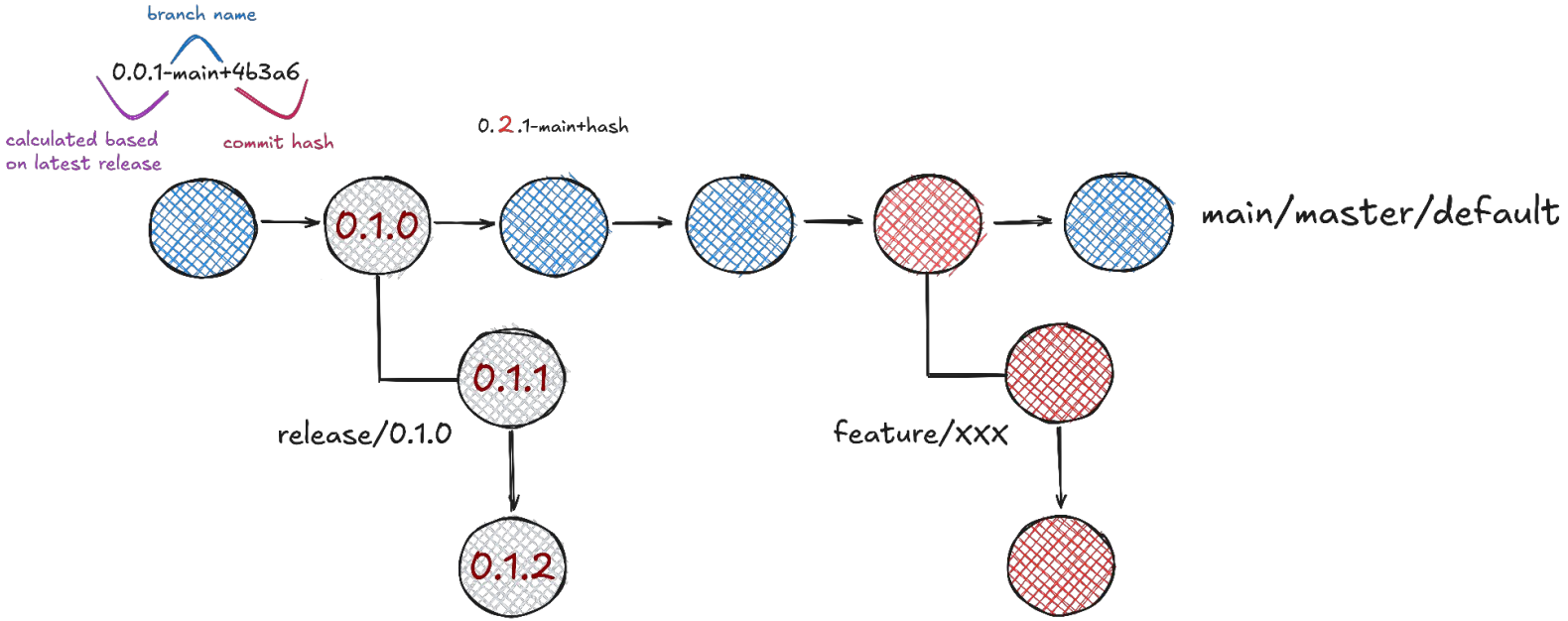
```
plugins {
    id("com.akuleshov7.vercraft.plugin-gradle") version("0.5.0")
}
```

```
$ ./gradlew build
>> VerCrafted: 0.6.2-main+c44cd
```

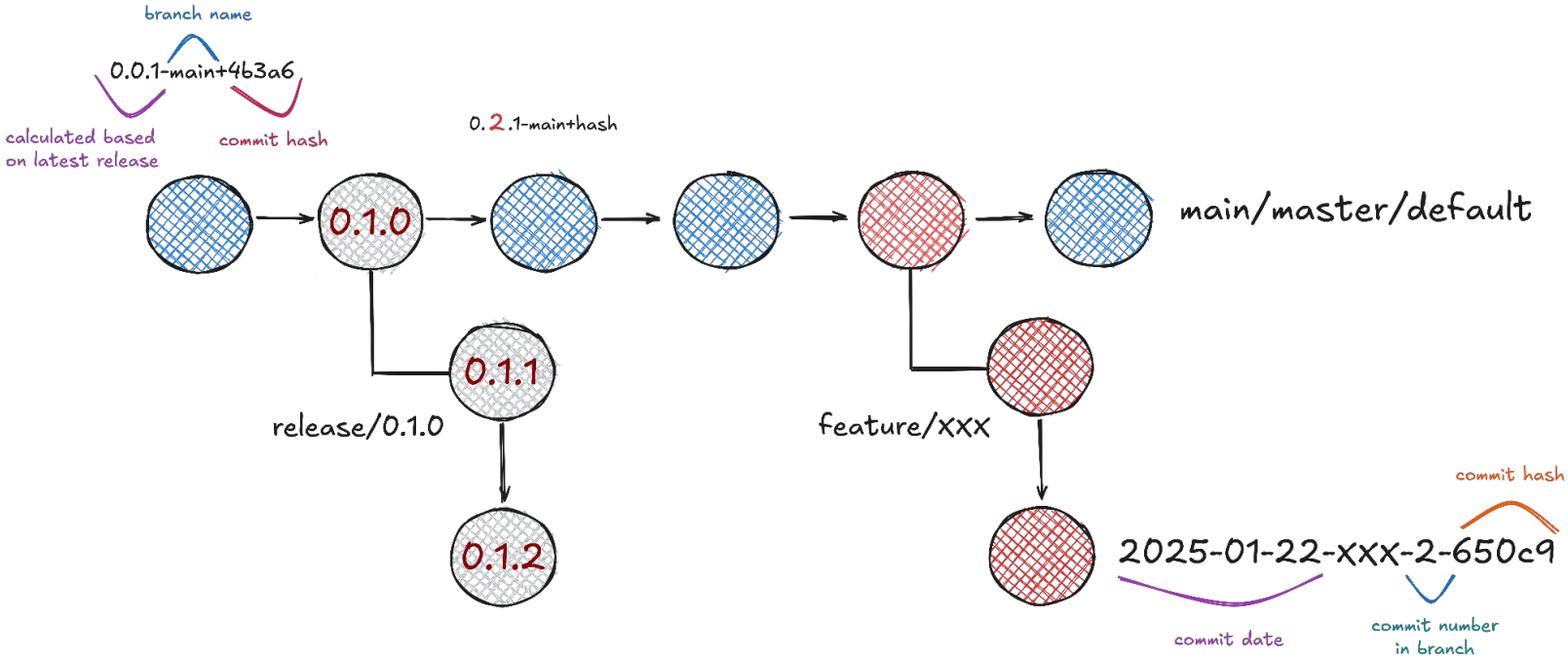
gitVersion task

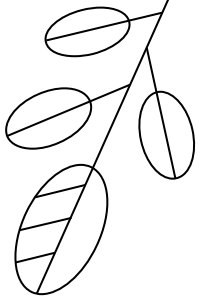


gitVersion task



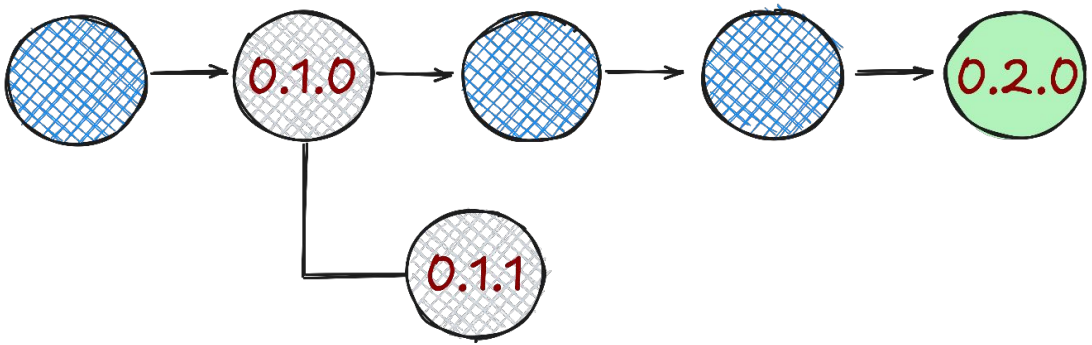
gitVersion task



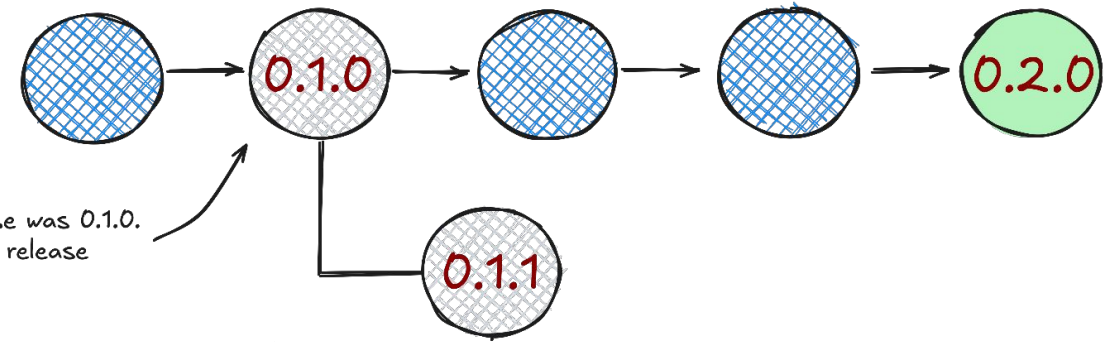


А чтобы направить людей на путь
истинный: **makeRelease**

makeRelease task

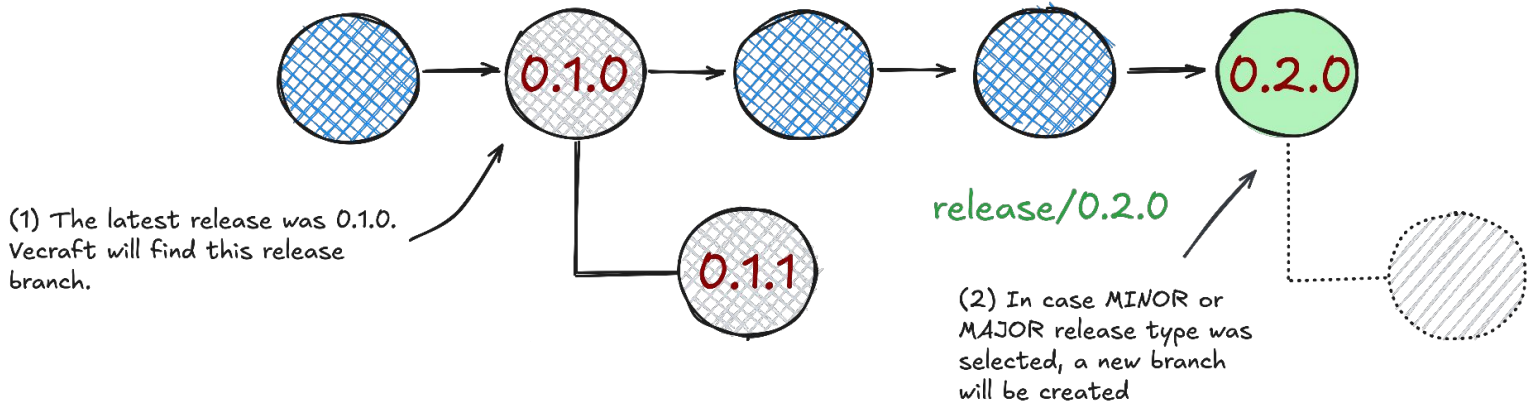


makeRelease task

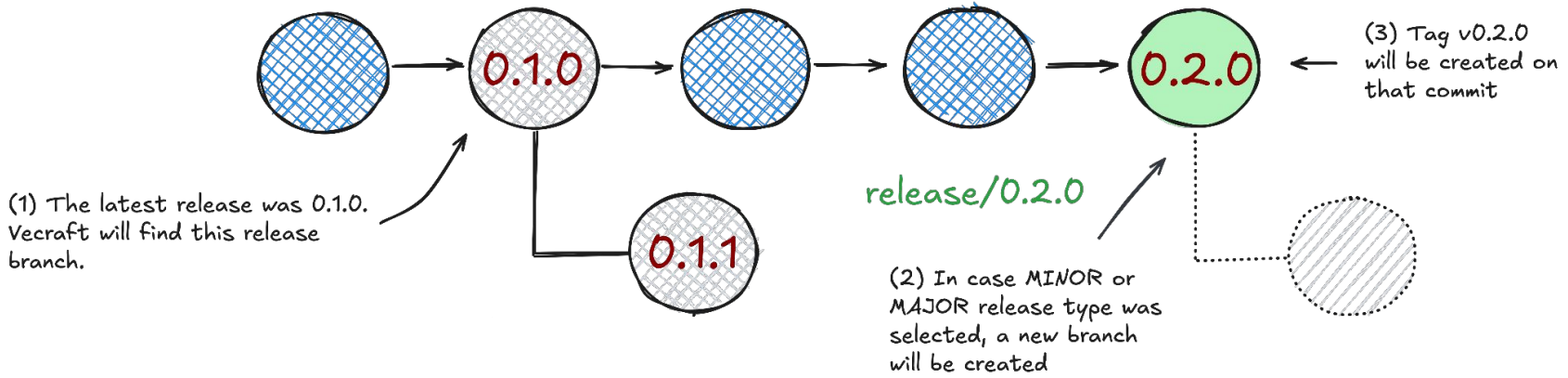


(1) The latest release was 0.1.0. Vecraft will find this release branch.

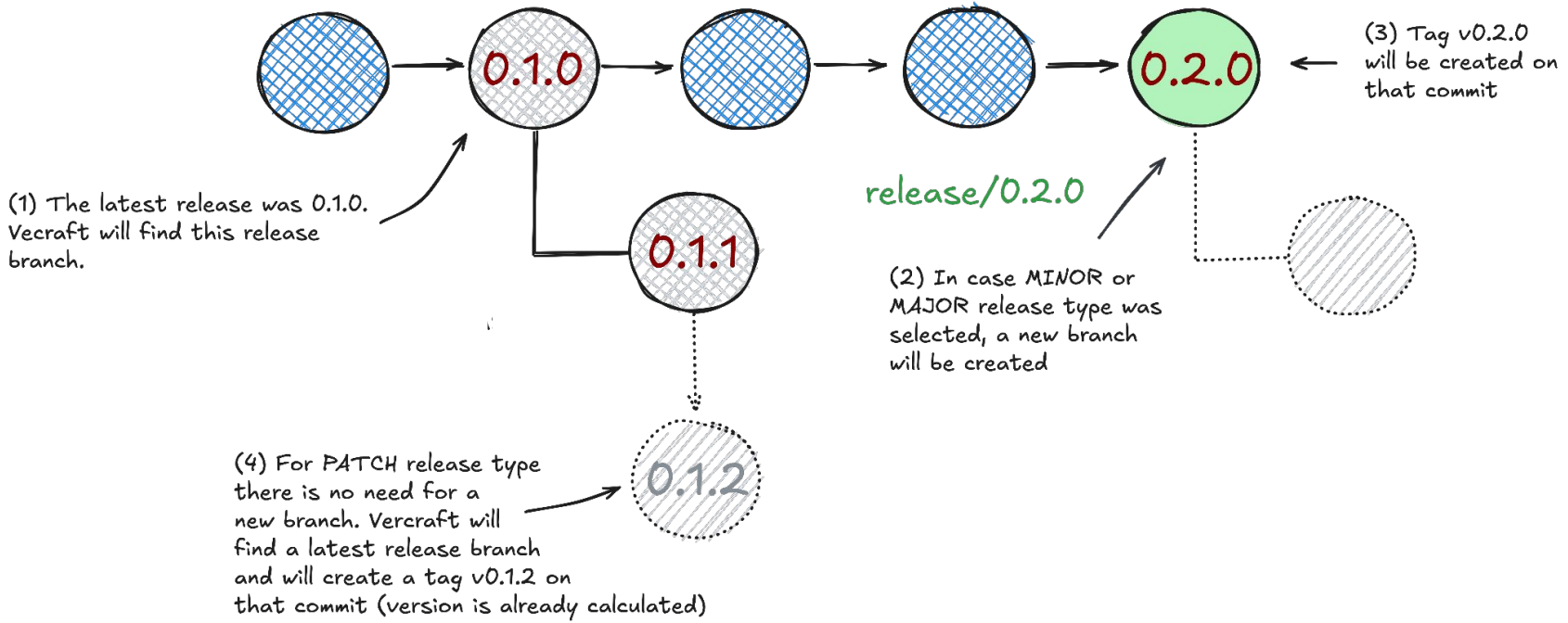
makeRelease task

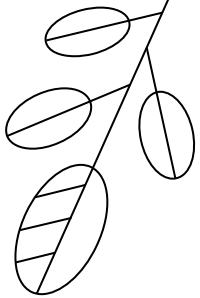


makeRelease task



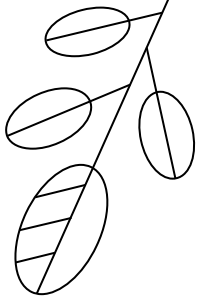
makeRelease task





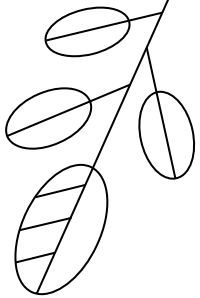
VerCraft

- Взял конечно же **JGit**



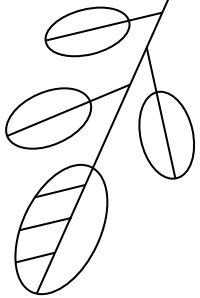
VerCraft

- Взял конечно же **JGit**
- Вдохновился **reckon’ом** и **nebul’ой**



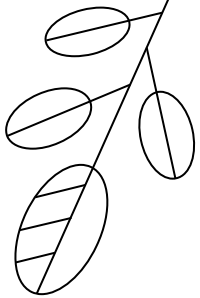
VerCraft

- Взял конечно же **JGit**
- Вдохновился **reckon’ом** и **nebul’ой**
- Версия фиксируется для build системы внутри



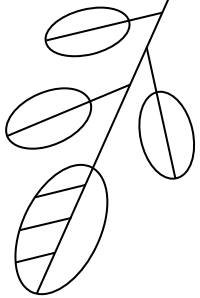
VerCraft

- Взял конечно же **JGit**
- Вдохновился **reckon’ом** и **nebul’ой**
- Версия фиксируется для build системы внутри
- Зависит от релизных бранчей, делает релизы



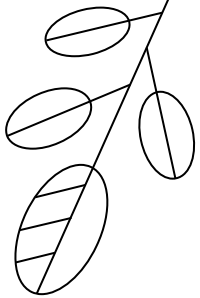
VerCraft

- Взял конечно же **JGit**
- Вдохновился **reckon’ом** и **nebul’ой**
- Версия фиксируется для build системы внутри
- Зависит от релизных бранчей, делает релизы
- Вышла пара версий, есть пользователи в PROD



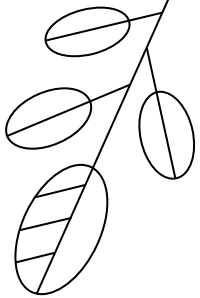
Выводы

- GitHub Flow + reckon - идеальный вариант для опенсорса



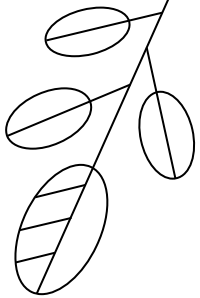
Выводы

- GitHub Flow + reckon - идеальный вариант для опенсорса
- GitLab Flow + зависимость от веток - для enterprise



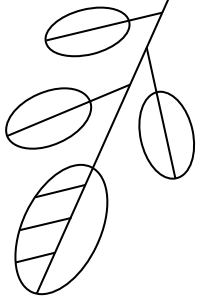
Выводы

- GitHub Flow + reckon - идеальный вариант для опенсорса
- GitLab Flow + зависимость от веток - для enterprise
- Trunk Based + свои велосипеды, если у вас много денег



Выводы

- GitHub Flow + reckon - идеальный вариант для опенсорса
- GitLab Flow + зависимость от веток - для enterprise
- Trunk Based + свои велосипеды, если у вас много денег
- Поиграйте с моим VerCraft’ом в реальной разработке



Выводы

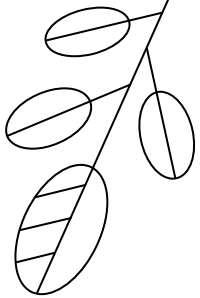
- GitHub Flow + reckon - идеальный вариант для опенсорса
- GitLab Flow + зависимость от веток - для enterprise
- Trunk Based + свои велосипеды, если у вас много денег
- Поиграйте с моим VerCraft’ом в реальной разработке
- Не боимся писать свои идеальные автоматизации с jGit

Спасибо!

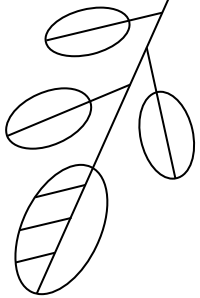
SourceCraft
от Яндекса

GitHub

Мой TG-канал



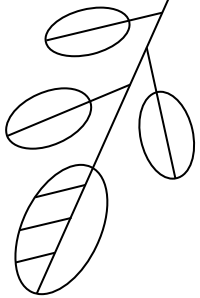
Секретный бонус!



Приколы нашего городка

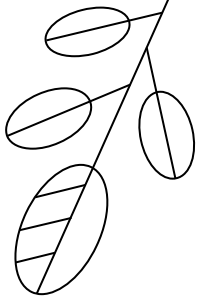
Detached commits

```
val branchName = config.checkoutBranch
    ?.value
    ?: System.getenv(GITLAB_BRANCH_REF)
    ?: System.getenv(GITHUB_HEAD_REF)
    ?: System.getenv(BITBUCKET_BRANCH)
    ?: System.getenv(VERCRAFT_BRANCH)
```



JGit и Kotlin

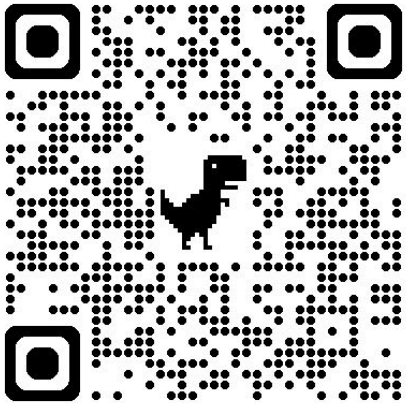
```
Git.open(gitPath).use { git →
    git.branchList()
        .setListMode(REMOTE)
        .call()

    git.branchCreate()
      .setName("release/${newVersion.semVerForNewBranch()}")
      .call()
}
```



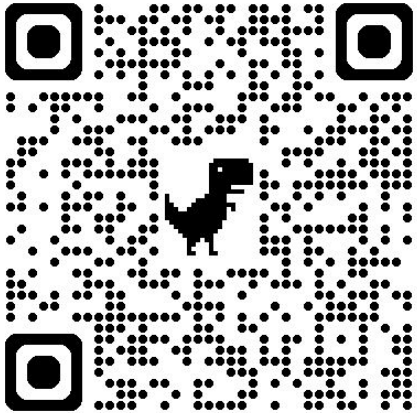
Синтетические коммиты

```
private val headCommit = RevWalk(repo).use {
    val realHeadOrMergeCommit = it.parseCommit(repo.resolve("HEAD"))
    // if it has two parents - then it is a merge commit
    // we also need to check that it is not present in the currentCheckoutBranch
    // (meaning, that it is a synthetic commit)
    if (realHeadOrMergeCommit.parentCount > 1 &&
            !currentCheckoutBranch.gitLog.contains(realHeadOrMergeCommit)) {
        // parent == 0 → commit from target branch (usually main)
        // parent == 1 → commit from source branch
        realHeadOrMergeCommit.getParent(1)
    } else {
        realHeadOrMergeCommit
    }
}
```

Спасибо!



GitHub



TG-канал