KGiSL

KG Information Systems Private Limited

Git Flow - Insurance Delivery

ENDLESS TECHNOLOGY TRANSFORMATION BUSINESS

01-Nov-2018

Agenda

DVCS | GIT

KGISL GSS DEVOPS CLUB

Distributed Version Control System (DVCS)

- In software development, distributed version control (also known as distributed revision control) is a form of version control where the complete codebase - including its full history - is mirrored on every developer's computer.
- This allows branching and merging to be managed automatically, increases speeds of most operations (except for pushing and pulling), improves the ability to work offline, and does not rely on a single location for backups

DVCS - GIT

- Git is a free and open source distributed version control system (DVCS)
- It supports for distributed, non-linear workflows (*thousands of parallel branches*)
- Speed & Simple design
- Extended RESTful API support
- If a central server crashes and all data is lost, any remote copy can be designated the official copy

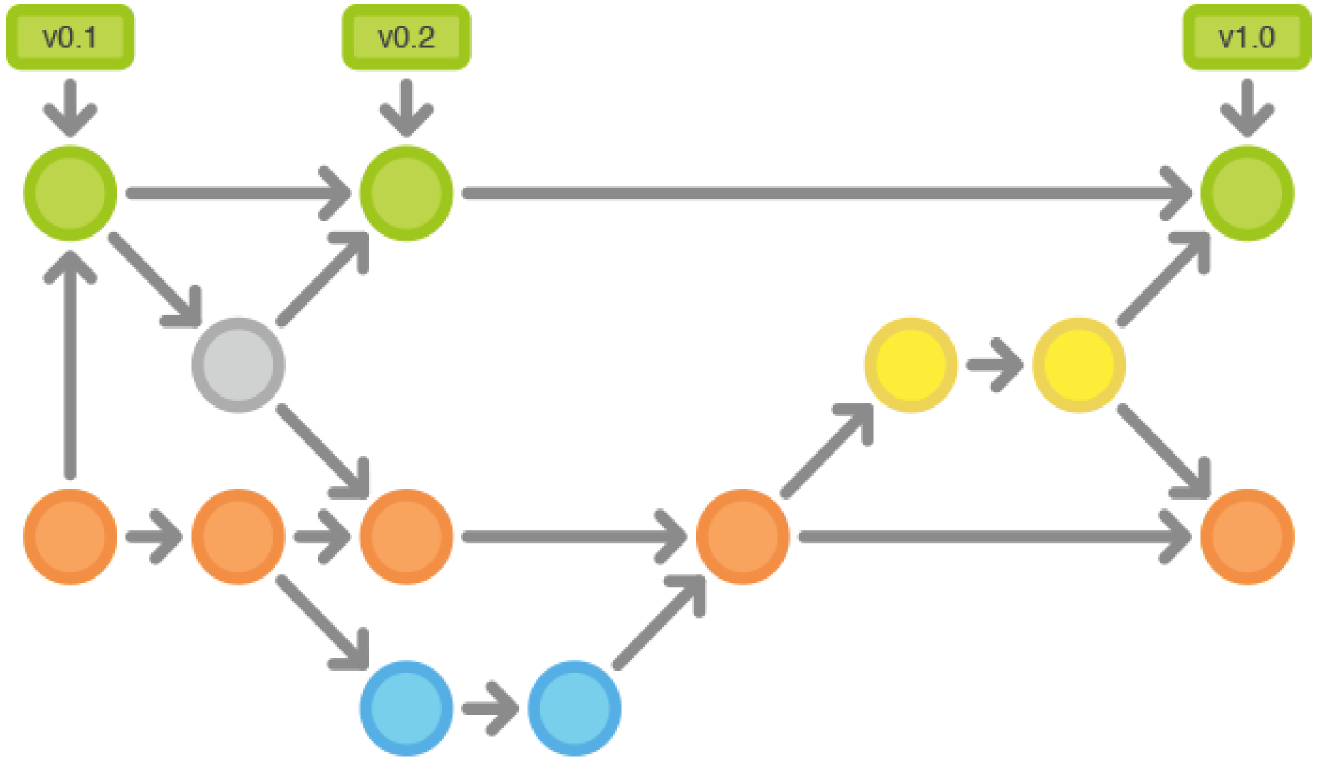
Agenda

KGiSL

Basic Git Flow

KGISL GSS DEVOPS CLUB

How Git Flow Works

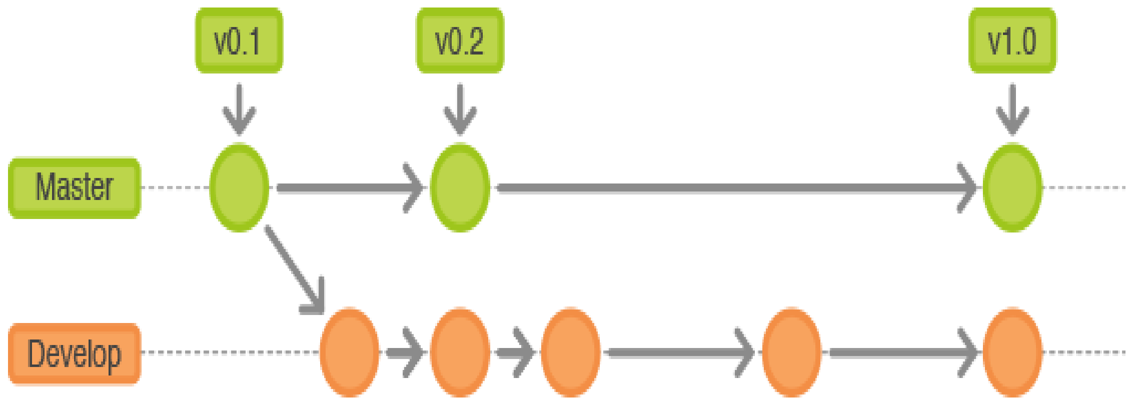


- The Git Flow workflow uses a central repository as the communication hub for all developers.
- Developers work locally and push branches to the central repo.

Historical Branches

- Instead of a single ***master*** branch, this workflow uses two branches to record the history of the project.
- The master branch stores the official release history, and the ***develop*** branch serves as an integration branch for features.
- It’s also convenient to tag all commits in the ***master*** branch with a version number.
- The rest of this workflow revolves around the distinction between these two branches.

Historical Branches



Best Practices for Historical Branches:

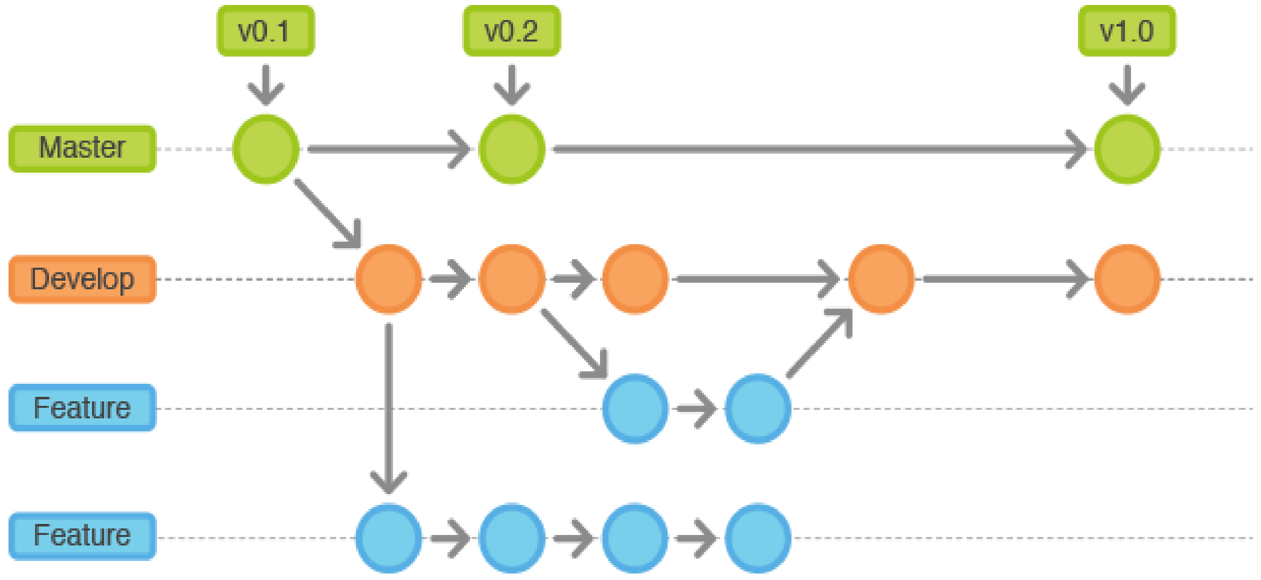
- ***master*** branch must be produced for developers
- Branch naming convention: should be ***master, develop***
- *Semantic Versioning should be mandatory on* ***tagging***

Feature Branches

- Each new feature should reside in its own branch, which can be pushed to the central repository for backup/collaboration.
- But, instead of branching off of master, feature branches use ***develop*** as their parent branch.
- When a feature is complete, it gets merged back into ***develop***.
- Features should never interact directly with ***master***.

Feature Branches

KGISL



Best Practices for Feature Branches:

- May branch off: ***develop***
- Must merge back into: ***develop***
- Branch naming convention: anything except ***master, develop, release-*, or hotfix-****

KGISL GSS DEVOPS CLUB

Release Branches

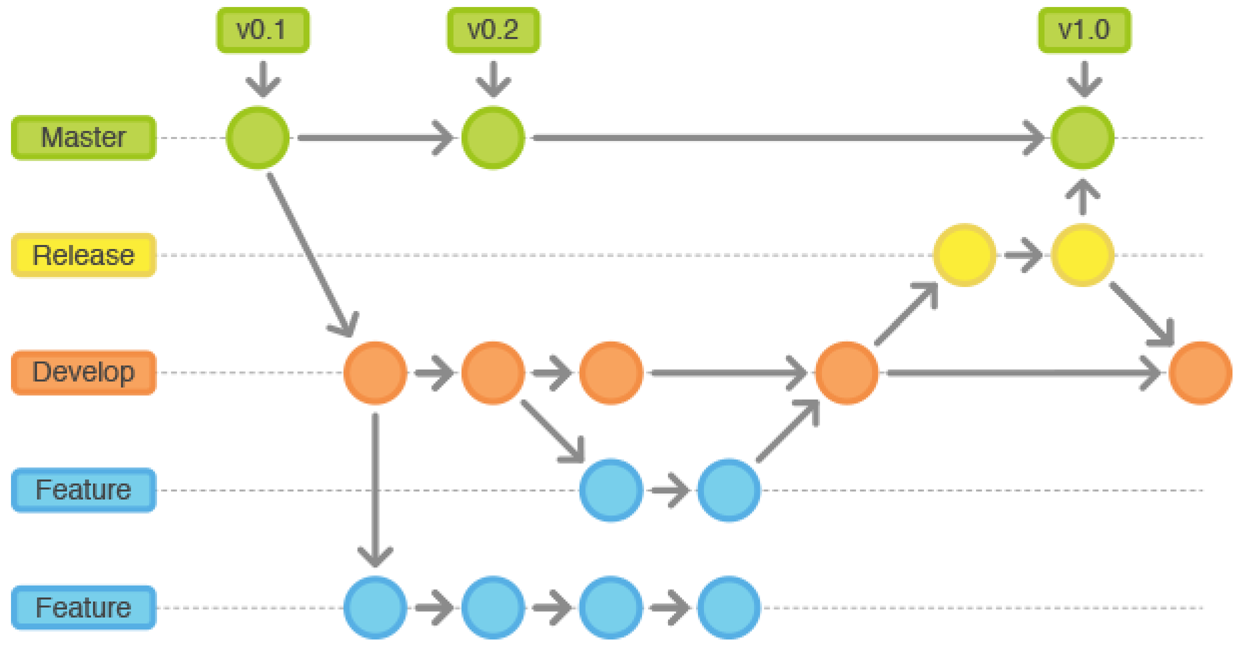
KGiSL

- Once develop has acquired enough features for a release (or a predetermined release date is approaching), you fork a ***release*** branch off of ***develop***.
- Creating this branch starts the next release cycle, so no new features can be added after this point-only bug fixes, documentation generation, and other release-oriented tasks should go in this branch.
- Once it's ready to ship, the release gets merged into ***master*** and tagged with a version number. In addition, it should be merged back into ***develop***, which may have progressed since the release was initiated.
- Using a dedicated branch to prepare releases makes it possible for one team to polish the current release while another team continues working on features for the next release.
- It also creates well-defined phases of development (e.g., it's easy to say, "this week we're preparing for version 4.0" and to actually see it in the structure of the repository).

KGISL GSS DEVOPS CLUB

Release Branches

KGiSL



Best Practices for Release Branches:

- May branch off: ***develop***
- Must merge back into: ***develop*** and ***master***
- Tag: increment major or minor number
- Branch naming convention: ***release-* or release/****

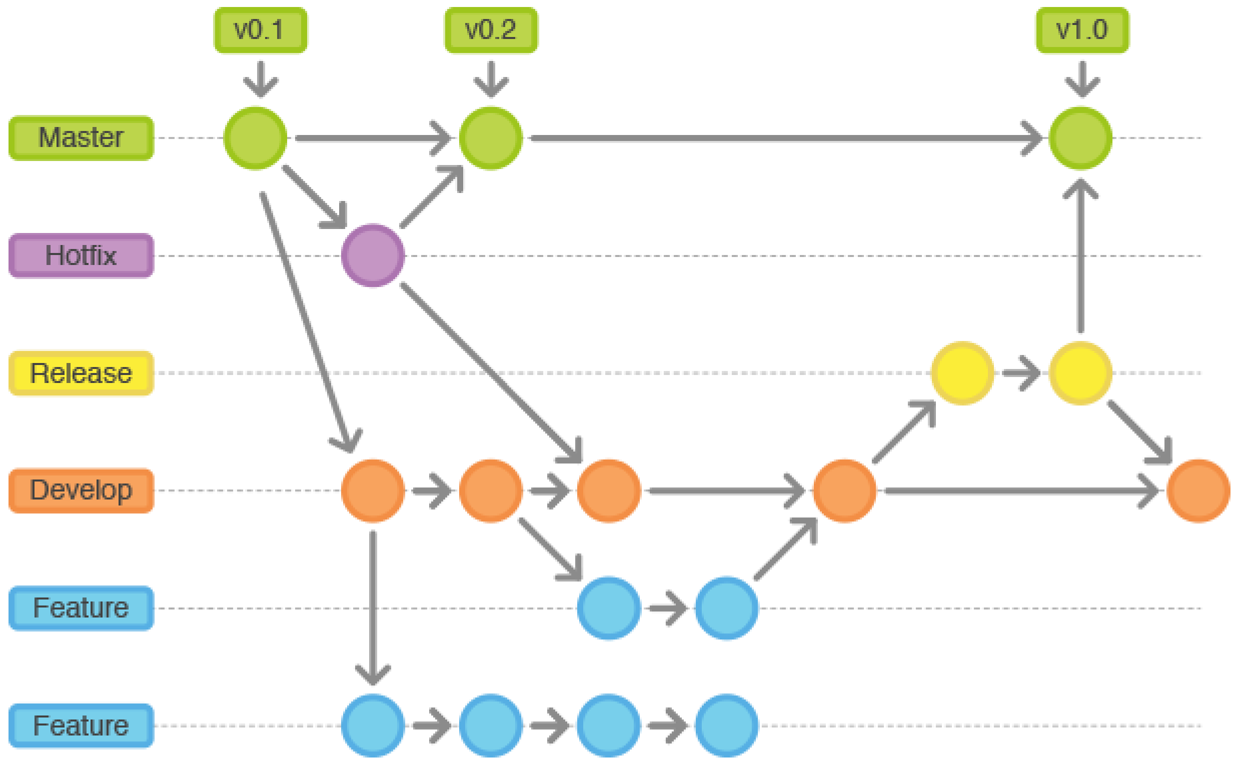
KGISL GSS DEVOPS CLUB

Maintenance Branches

- Maintenance or "***hotfix***" branches are used to quickly patch production releases.
- This is the only branch that should fork directly off of master.
- As soon as the fix is complete, it should be merged into both ***master*** and ***develop*** (or the current release branch), and master should be tagged with an updated version number.
- Having a dedicated line of development for bug fixes lets your team address issues without interrupting the rest of the workflow or waiting for the next release cycle.
- You can think of maintenance branches as ad hoc release branches that work directly with master.

Maintenance Branches

KGiSL



Best Practices for Maintenance Branches:

- May branch off: ***master***
- Must merge back into: ***master*** and ***develop***
- Tag: increment patch number
- Branch naming convention: ***hotfix-* or hotfix/****

KGISL GSS DEVOPS CLUB

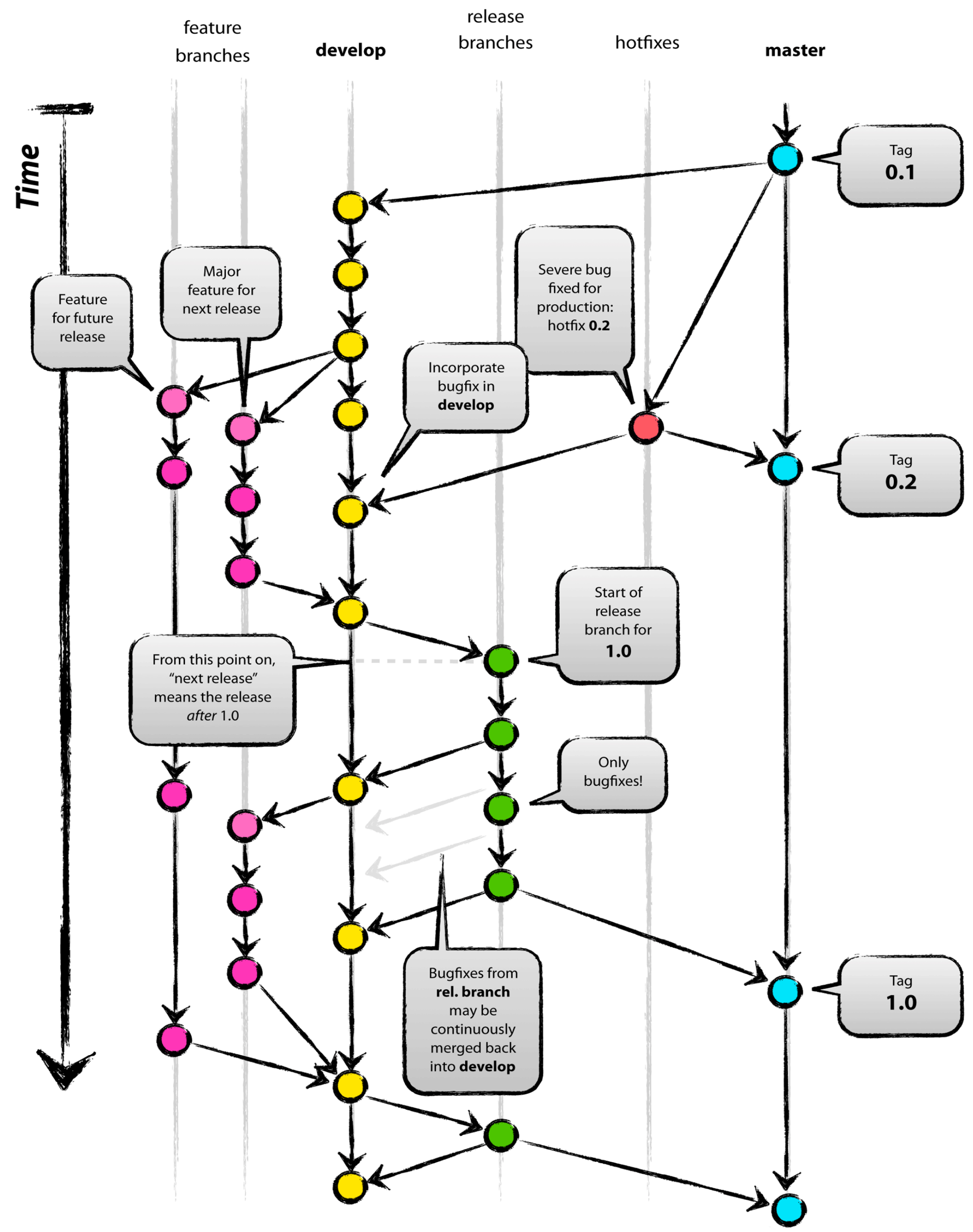
Agenda

KGiSL

Git Workflow - Lifecycle

KGISL GSS DEVOPS CLUB

Basic Git Flow - Lifecycle



feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Major feature for next release
Feature for future release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

KGISL GSS DEVOPS CLUB

Agenda

KGiSL

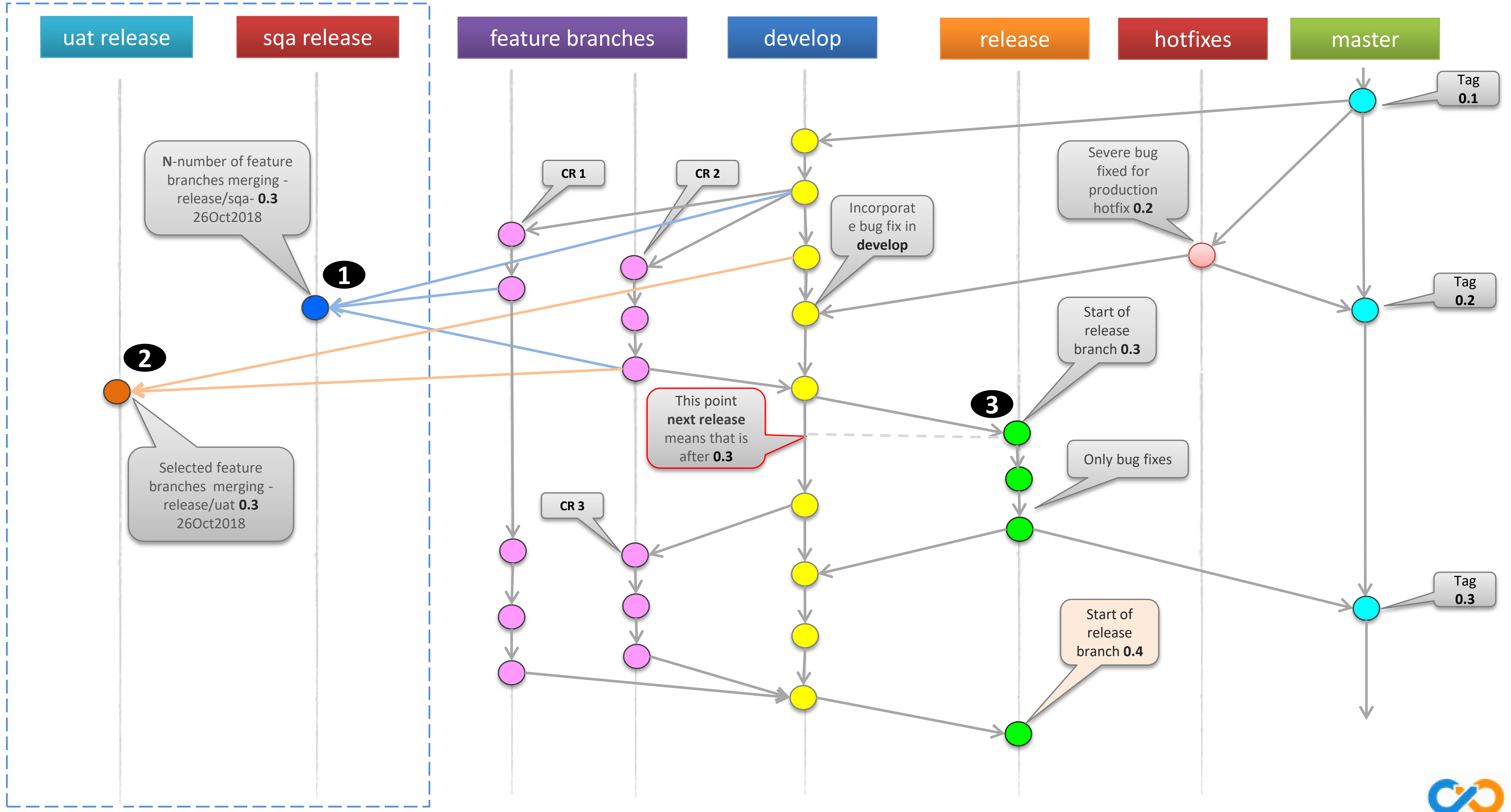
Custom Built – Git Flow

KGISL GSS DEVOPS CLUB

Why Custom built Git flow?

- Basic Git flow emphasis if **release candidate** initiated that should be prioritized to production – failing pick & release
- Insurance releases required seamless **integration** on SQA & UAT release candidates.
- Feature/Bug branches will be **picked & released** in SQA & UAT environment as per testers & users requirements.
- To implement Semantic Versioning policy.

Custom Built – Git Flow


uat release
sqa release
feature branches
develop
release
hotfixes
master
Tag 0.1
Tag 0.2
Tag 0.3
CR 1
CR 2
CR 3
N-number of feature branches merging - release/sqa- 0.3 26Oct2018
1
2
3
Incorporate bug fix in develop
Severe bug fixed for production hotfix 0.2
Start of release branch 0.3
This point next release means that is after 0.3
Only bug fixes
Selected feature branches merging - release/uat 0.3 26Oct2018
Start of release branch 0.4

Integration Hub

KGISL GSS DEVOPS CLUB

Integration Hub – SQA Release

KGiSL

- SQA **release** branch will be created from develop branch
- Selected ***feature*** branches will be merged into the SQA release channel – example : ***release/sqa-0.3-26Oct2018-02-rhbi***
- Incremental semantic versioning to be followed from SQA release onwards – **to stick with dedicate release version**
- The *SQA release* branch will be deployed for offshore manual & automated testing
- In any case, developers must work in respective feature branch always
- The same SQA release loop will be followed for next SQA deployment

KGISL GSS DEVOPS CLUB

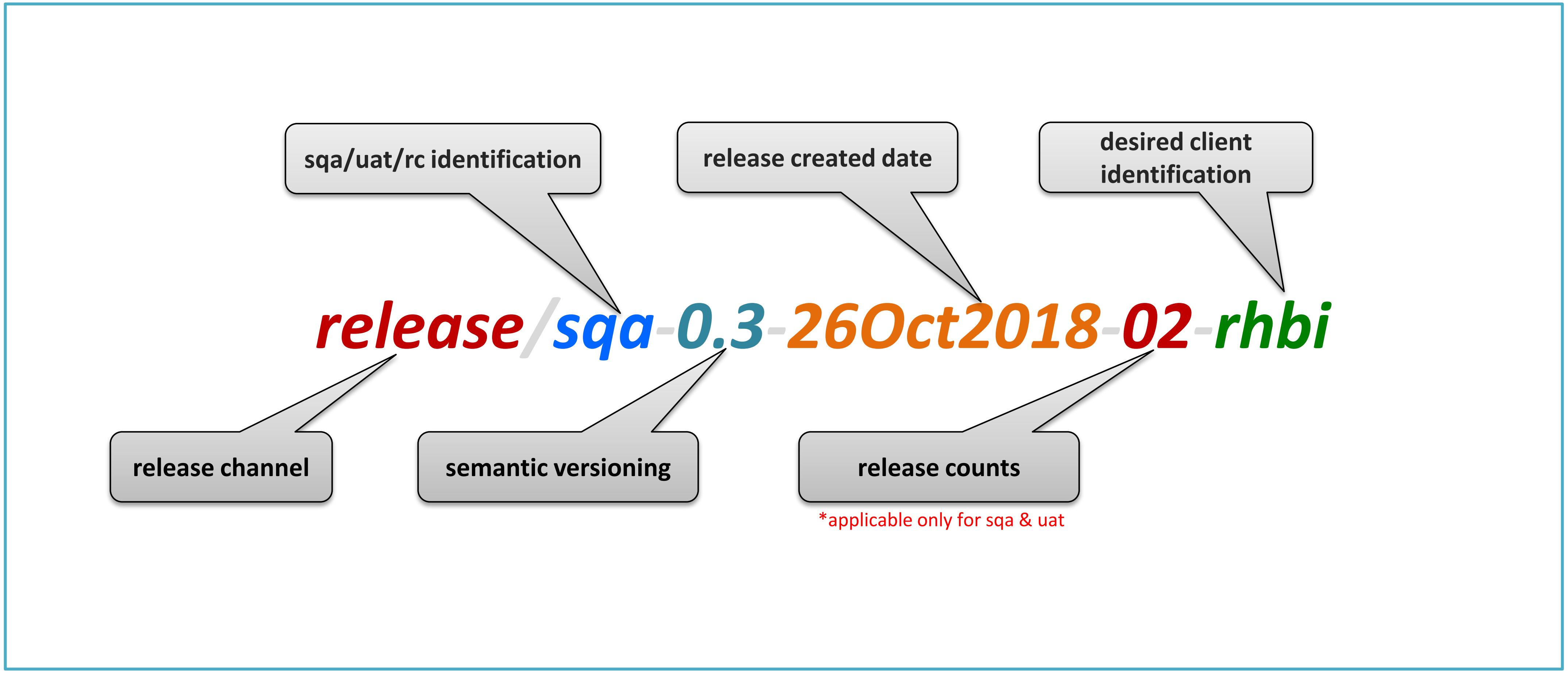
Integration Hub – UAT Release

- On SQA completion, UAT release branch will be created from develop branch
- Selected feature branches will be merged into the UAT release channel – example : ***release/uat-0.3-26Oct2018-02-rhbi***
- Same SQA release versioning number to be used for UAT release practice – **to stick with dedicate release version (final RC)**
- The *uat release* branch will be released for offshore and onsite testing scope
- In any case, developers must work in respective feature branch always
- The same UAT release loop will be followed for next UAT deployment

Regression Release (RC)

- User signed-off ***feature*** branches will be merged back into develop branch for final Release Candidate(RC) branch commencement
- On Impact Analysis completion, the final RC will be created from the merging point of develop branch – example : ***release/rc-0.3-28Oct2018-rhbi***
- The same SQA & UAT release versioning number will be used for final RC branch – **to stick with dedicate release version (as final RC)**
- From this point on '*next release*' means the release after ***rc-0.3***
- After this point-only bug fixes, documentation generation, and other release-oriented tasks should go in this branch
- The ***final RC*** branch will be released for offshore and onsite regression testing scope
- Once it's ready to ship, the release gets merged into master and tagged with a version number

Release Branch Naming Convention


sqa/uat/rc identification
release created date
desired client identification
release/sqa-0.3-26Oct2018-02-rhbi
release channel
semantic versioning
release counts
*applicable only for sqa & uat

Agenda

Git Workflow– Key Principles

Key Principles

- Do not mess with the master.
- Why is the master so important to not mess with? One word: the master branch is deployable. It is your production code, ready to roll out into the world. The master branch is meant to be stable
- Software to never, ever push anything to master that is not tested , or that breaks the build
- The master branch stores the official release history
- The develop branch serves as an integration branch for features
- Feature branches must use develop as their parent branch
- Features should never interact directly with master
- Review others’ code independently.

Key Principles

KGiSL

- Using a dedicated branch to prepare releases
- Maintenance branches as ad hoc release branches that work directly with master
- Hotfix/Maintenance should be merged into both master and develop (and the current release branch if any)
- And master should be tagged with an updated semantic version number
- Should have ‘documented/defined’ branching strategy
- Do regularly review and delete ‘dead branches’

KGiSL

KG Information Systems Private Limited

THANK YOU

www.kgisl.com

ENDLESS TECHNOLOGY TRANSFORMATION BUSINESS