GIT BRANCHING MODEL
FOR EFFICIENT DEVELOPMENT

LEMİ ORHAN ERGİN
PRINCIPAL SOFTWARE ENGINEER @ SONY

THIS DOCUMENT IS PREPARED FOR THE TECHNICAL TALK ON 14TH OF NOVEMBER, 2012 AT SONY

WHY GIT IS GOOD ?

CHEAP LOCAL BRANCHING

EVERYTHING IS LOCAL

GIT IS FAST

GIT IS SMALL

THE STAGING AREA

DISTRIBUTED

ANY WORKFLOW

GITHUB

EASY TO LEARN

GIT IS THE NEW STANDARD

HUGE COMMUNITY

Changed the rules
2,538,360 people
4,315,025 repositories
Raised $100 million on July'12

CONCEPTS

BRANCH

SEPARATE LINE OF WORK

PUBLIC BRANCH

A PUBLIC BRANCH IS ONE THAT MORE THAN ONE PERSON PULLS FROM

TOPICAL (FEATURE) BRANCH

PRIVATE BRANCH THAT YOU ALONE ARE USING, AND WILL NOT EXPOSED IN THE PUBLIC REPOSITORY

TRACKING BRANCH

LOCAL BRANCH THAT KNOWS WHERE ITS REMOTE IS, AND THAT CAN PUSH TO AND PULL FROM THAT REMOTE

MERGING AND BRANCHING IN GIT

GIT HAS CHANGED THE WAY DEVELOPERS THINK OF MERGING AND BRANCHING

WITH GIT, MERGING AND BRANCHING ARE EXTREMELY CHEAP AND SIMPLE, AND THEY ARE CONSIDERED ONE OF THE CORE PARTS OF YOUR DAILY WORKFLOW

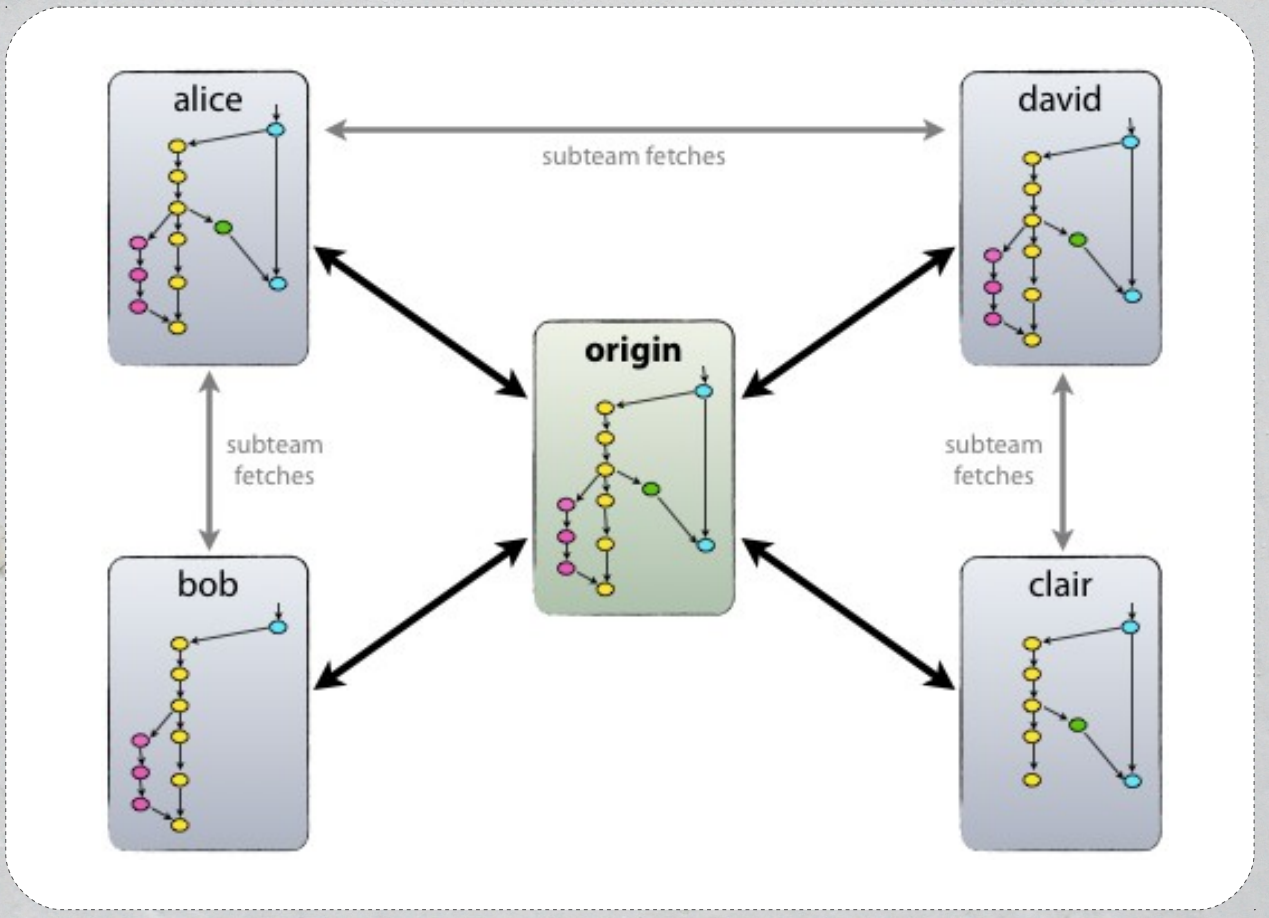
WORKFLOW

VERSION CONTROL WORKFLOW HANDLES FEATURES, BUGS AND HOT FIXES ON YOUR CODEBASE AIMED TO RUN ON MULTIPLE ENVIRONMENTS WHILE KEEPING A CLEAN AND SANE HISTORY

GIT FOR DESIGNING WORKFLOWS

IT IS REALLY “A TOOL FOR DESIGNING VCS WORKFLOWS”
RATHER THAN A VERSION CONTROL SYSTEM ITSELF.
OR, AS LINUS WOULD SAY, “GIT IS JUST A STUPID CONTENT TRACKER”

REPOSITORY MANAGEMENT IN GIT:
DECENTRALIZED BUT CENTRALIZED



MAIN BRANCHES

WE CONSIDER **ORIGIN/PRODUCTION** TO BE THE MAIN BRANCH WHERE THE SOURCE CODE OF HEAD ALWAYS REFLECTS A **PRODUCTION-READY STATE**.

WE CONSIDER **ORIGIN/MASTER** TO BE THE MAIN BRANCH WHERE THE SOURCE CODE OF HEAD ALWAYS REFLECTS A STATE WITH THE **LATEST DELIVERED DEVELOPMENT** CHANGES FOR THE NEXT RELEASE. SOME WOULD CALL THIS THE “INTEGRATION BRANCH”. THIS IS WHERE ANY AUTOMATIC NIGHTLY BUILDS ARE BUILT FROM.

MORE MAIN BRANCHES

WE COULD ALSO CONSIDER ORIGIN/STAGING TO BE THE MAIN BRANCH WHERE THE SOURCE CODE OF HEAD ALWAYS REFLECTS A STATE WITH LATEST CODE CHANGES AND BUG FIXES FOR THE STAGING ENVIRONMENT.

WITH THIS LOGIC, YOU CAN DEFINE PERSISTENT BRANCHES FOR EACH ENVIRONMENT. THAT'S UP YOUR NEEDS.

RELEASE

EACH TIME WHEN CHANGES ARE MERGED BACK INTO PRODUCTION,
THIS IS A NEW PRODUCTION RELEASE BY DEFINITION

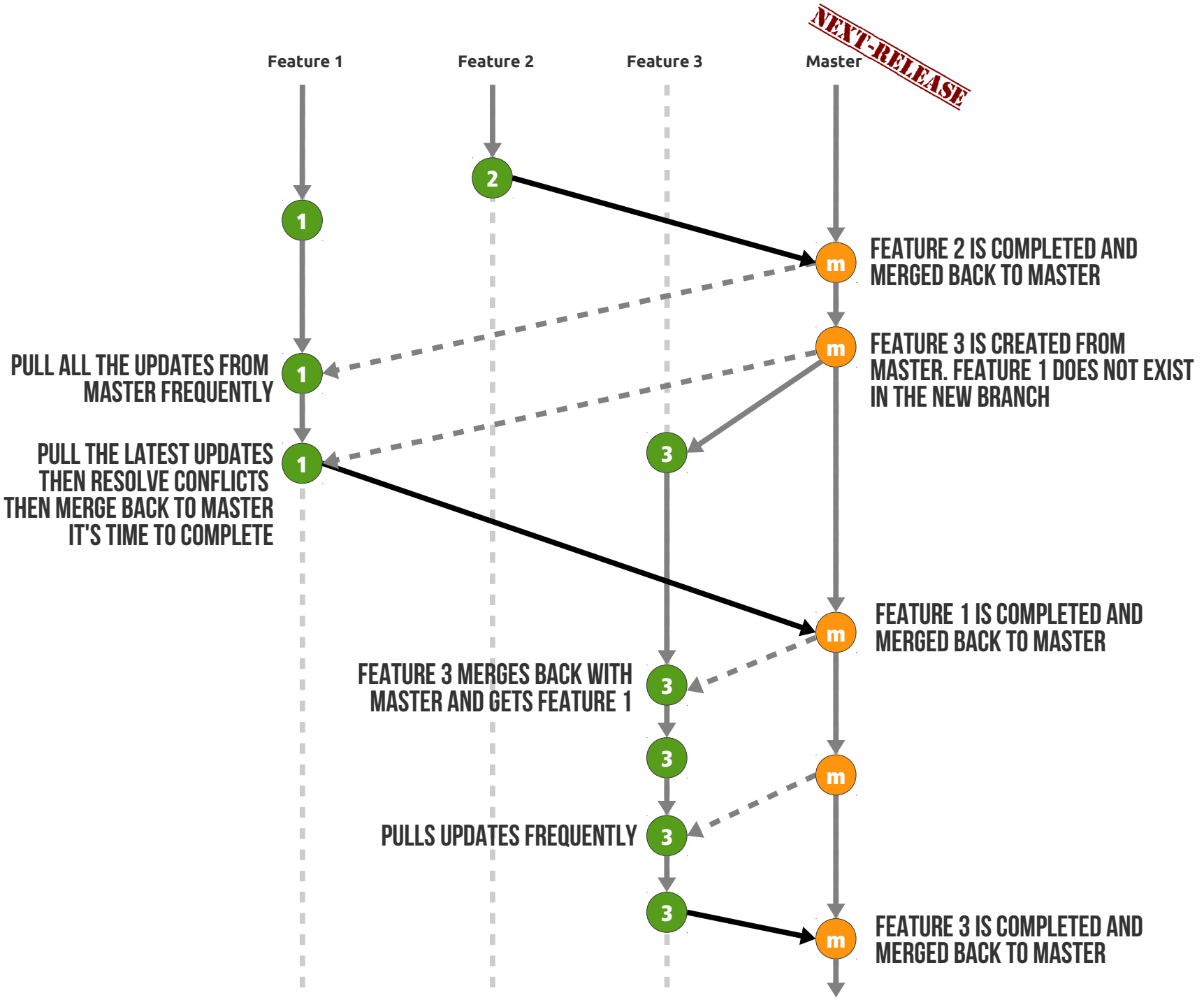
SUPPORTING BRANCHES

UNLIKE THE MAIN BRANCHES, THESE BRANCHES ALWAYS HAVE A **LIMITED LIFE TIME** SINCE THEY WILL BE REMOVED EVENTUALLY.

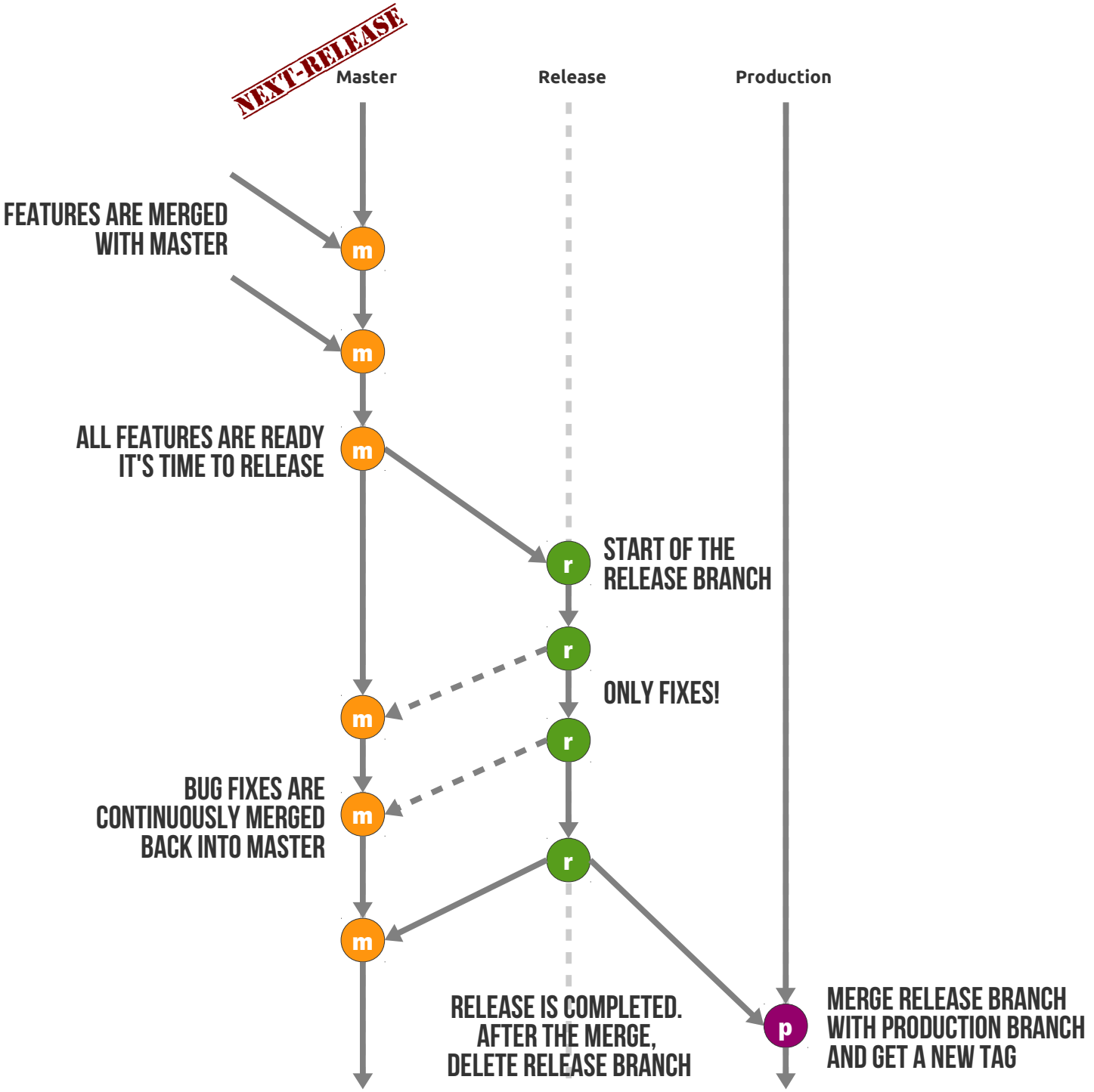
EACH OF THESE BRANCHES HAVE A **SPECIFIC PURPOSE** AND ARE BOUND TO **STRICT RULES** AS TO WHICH BRANCHES MAY BE THEIR ORIGINATING BRANCH AND WHICH BRANCHES MUST BE THEIR MERGE TARGETS.

FEATURE BRANCHES
HOTFIX BRANCHES
RELEASE BRANCHES

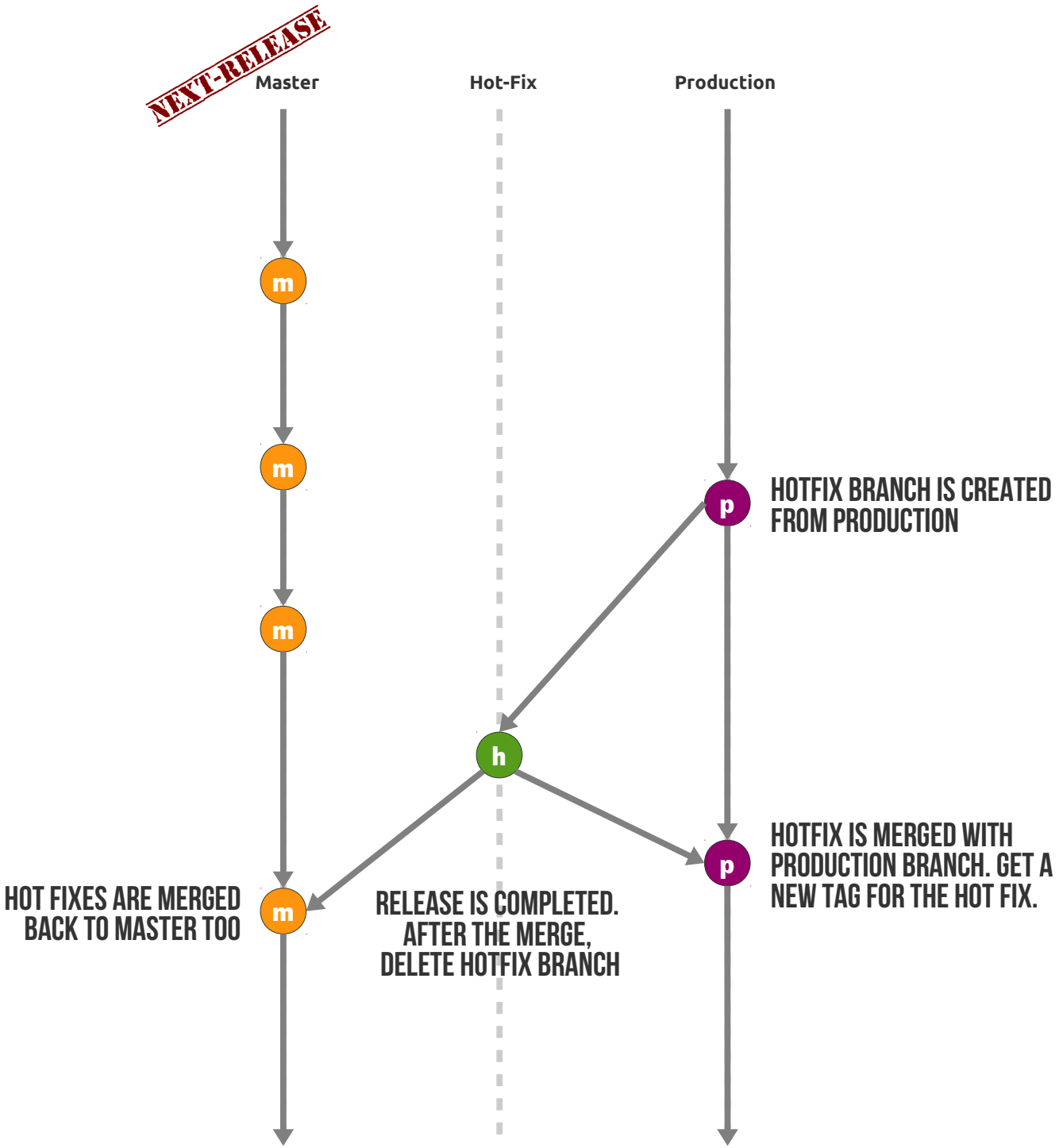
FEATURE DEVELOPMENT



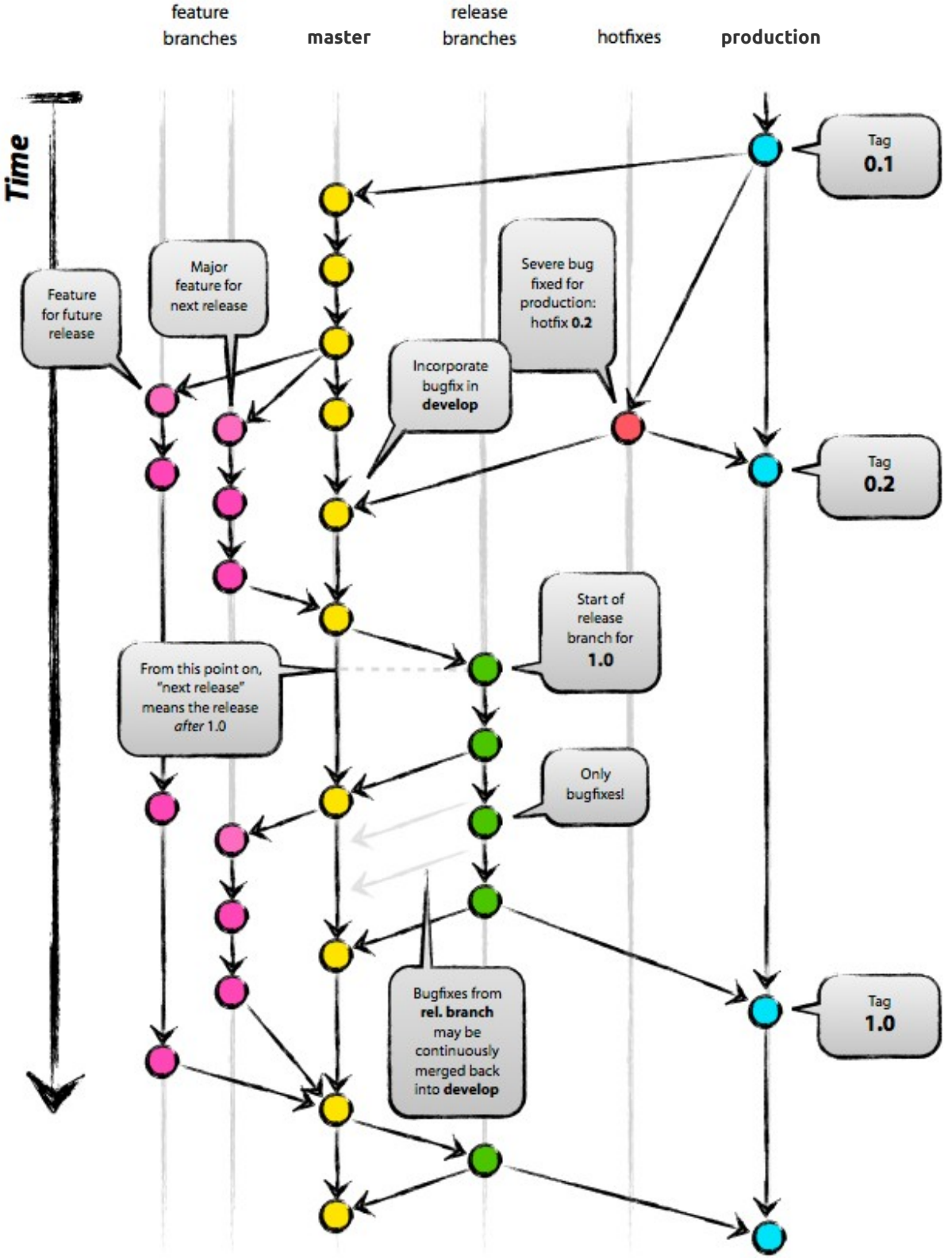
RELEASE MANAGEMENT



HOT-FIXES IN PRODUCTION



ALL FLOWS IN BRANCHING MODEL



ORIGINAL GRAPH IS FROM "A SUCCESSFUL GIT BRANCHING MODEL" BY VINCENT DRIESSEN

IT'S TIME TO

IMPLEMENT

but how?

REBASE vs MERGE

AND THE DIFFERENCE IN KEEPING THE HISTORY (1)

MERGING BRINGS TWO LINES OF DEVELOPMENT TOGETHER WHILE PRESERVING THE ANCESTRY OF EACH COMMIT HISTORY.

IN CONTRAST, **REBASING** UNIFIES THE LINES OF DEVELOPMENT BY RE-WRITING CHANGES FROM THE SOURCE BRANCH SO THAT THEY APPEAR AS CHILDREN OF THE DESTINATION BRANCH – EFFECTIVELY PRETENDING THAT THOSE COMMITS WERE WRITTEN ON TOP OF THE DESTINATION BRANCH ALL ALONG.

REBASE REQUIRES THE COMMITS ON THE SOURCE BRANCH TO BE RE-WRITTEN, WHICH CHANGES THEIR CONTENT AND THEIR SHAS

REBASE vs MERGE

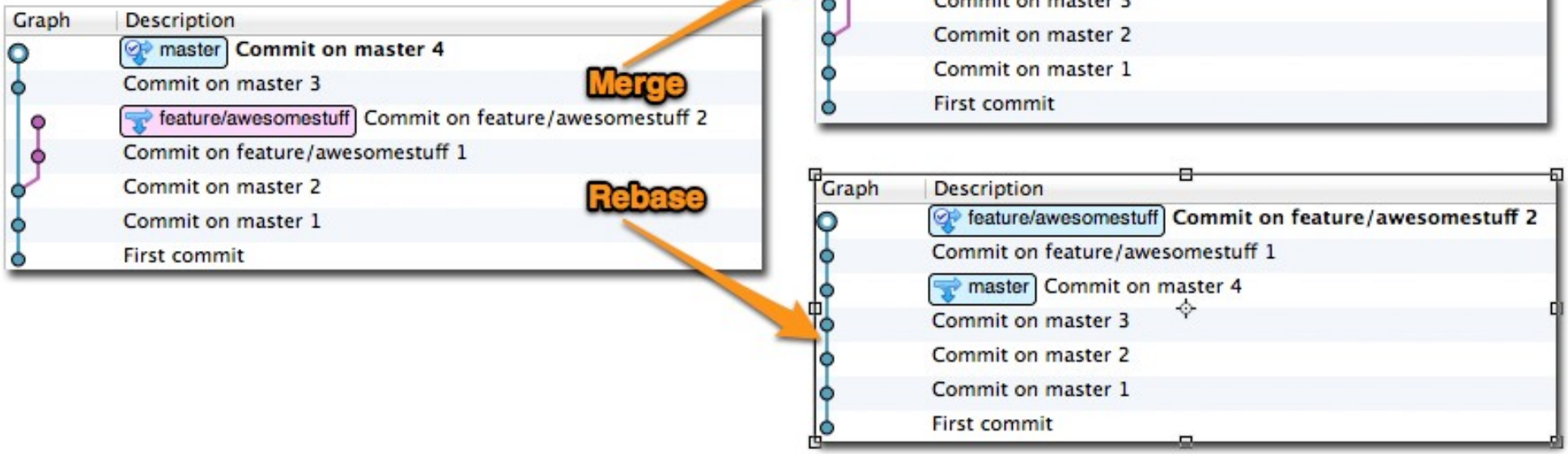
AND THE DIFFERENCE IN KEEPING THE HISTORY (2)

MERGING IS BETTER YOU ONLY HAVE ONE (OR FEW THRUSTED) COMMITTER AND YOU DON'T CARE MUCH ABOUT READING YOUR HISTORY.

REBASING MAKES YOU SURE THAT YOUR COMMITS GO ON TOP OF THE "PUBLIC" BRANCH

REBASE vs MERGE

AND WHAT IT MEANS IN MEANS OF HISTORY WRITING



HTTP://BLOG.SOURCETREEAPP.COM/FILES/2012/08/MERGEREBASE.PNG

MERGE PROS AND CONS

PROS

1. SIMPLE TO USE AND UNDERSTAND.
2. THE COMMITS ON THE SOURCE BRANCH REMAIN SEPARATE FROM OTHER BRANCH COMMITS, PROVIDED YOU DON'T PERFORM A FAST-FORWARD MERGE. (THIS SEPARATION CAN BE USEFUL IN THE CASE OF FEATURE BRANCHES, WHERE YOU MIGHT WANT TO TAKE A FEATURE AND MERGE IT INTO ANOTHER BRANCH LATER)
3. EXISTING COMMITS ON THE SOURCE BRANCH ARE UNCHANGED AND REMAIN VALID; IT DOESN'T MATTER IF THEY'VE BEEN SHARED WITH OTHERS.

CONS

1. IF THE NEED TO MERGE ARISES SIMPLY BECAUSE MULTIPLE PEOPLE ARE WORKING ON THE SAME BRANCH IN PARALLEL, THE MERGES DON'T SERVE ANY USEFUL HISTORIC PURPOSE AND CREATE CLUTTER.

REBASE PROS AND CONS

PROS

1. SIMPLIFIES YOUR HISTORY.
2. IS THE MOST INTUITIVE AND CLUTTER-FREE WAY TO COMBINE COMMITS FROM MULTIPLE DEVELOPERS IN A SHARED BRANCH

CONS

1. SLIGHTLY MORE COMPLEX, ESPECIALLY UNDER CONFLICT CONDITIONS. (EACH COMMIT IS REBASED IN ORDER, AND A CONFLICT WILL INTERRUPT THE PROCESS OF REBASING MULTIPLE COMMITS.)
2. REWRITING OF HISTORY HAS RAMIFICATIONS IF YOU'VE PREVIOUSLY PUSHED THOSE COMMITS ELSEWHERE. (YOU MAY PUSH COMMITS YOU MAY WANT TO REBASE LATER (AS A BACKUP) BUT ONLY IF IT'S TO A REMOTE BRANCH THAT ONLY YOU USE. IF ANYONE ELSE CHECKS OUT THAT BRANCH AND YOU LATER REBASE IT, IT'S GOING TO GET VERY CONFUSING.)

GOLDEN RULE

OF REBASING

NEVER EVER

REBASE A BRANCH THAT YOU PUSHED,
OR THAT YOU PULLED FROM ANOTHER PERSON

REBASE OR MERGE

USE WHICH STRATEGY WHEN

PUSH:

1. DON'T DO YOUR WORK ON THE PUBLIC BRANCH, USE FEATURE BRANCHES
2. WHEN MULTIPLE DEVELOPERS WORK ON A SHARED BRANCH, PUSH & REBASE YOUR OUTGOING COMMITS TO KEEP HISTORY CLEANER
3. TO RE-INTEGRATE A COMPLETED FEATURE BRANCH, USE MERGE (AND OPT-OUT OF FAST-FORWARD COMMITS IN GIT)

PULL:

1. TO BRING A FEATURE BRANCH UP TO DATE WITH ITS BASE BRANCH, PREFER REBASING YOUR FEATURE BRANCH ONTO THE LATEST BASE BRANCH IF YOU HAVEN'T PUSHED THIS BRANCH ANYWHERE YET, OR YOU KNOW FOR SURE THAT OTHER PEOPLE WILL NOT HAVE CHECKED OUT YOUR FEATURE BRANCH
2. OTHERWISE, MERGE THE LATEST BASE CHANGES INTO YOUR FEATURE BRANCH

FEATURE DEVELOPMENT

1. PULL TO UPDATE YOUR LOCAL MASTER
2. CHECK OUT A FEATURE BRANCH FROM MASTER
3. DO WORK IN YOUR FEATURE BRANCH, COMMITTING EARLY AND OFTEN
4. REBASE FREQUENTLY TO INCORPORATE UPSTREAM CHANGES
5. INTERACTIVE REBASE (SQUASH) YOUR COMMITS
6. MERGE YOUR CHANGES WITH MASTER
7. PUSH YOUR CHANGES TO THE UPSTREAM

1 PULL TO UPDATE YOUR LOCAL MASTER

```
git checkout master
git pull origin master
```

THIS SHOULD NEVER CREATE A MERGE COMMIT BECAUSE WE ARE NEVER WORKING DIRECTLY IN MASTER.
WHENEVER YOU PERFORM A PULL, MERGE OR REBASE, MAKE SURE THAT YOU RUN TESTS DIRECTLY AFTERWARDS.

2 CHECK OUT A FEATURE BRANCH FROM MASTER

```
git checkout -b feature-1185-add-commenting
```

CHECK OUT A FEATURE BRANCH NAMED WITH THE STORY ID AND A SHORT, DESCRIPTIVE TITLE.
THE ID ALLOWS US TO EASILY TRACK THIS BRANCH BACK TO THE STORY THAT SPAWNED IT. THE TITLE IS THERE TO GIVE US HUMANS A LITTLE HINT AS TO WHAT'S IN IT.

TIP

PULL = FETCH + MERGE
YOU MAY USE REBASE INSTEAD OF MERGE WITH THE PULL
"GIT PULL --REBASE <REMOTE BRANCH> <LOCAL BRANCH>"
"GIT CONFIG BRANCH.AUTOSETUPREBASE ALWAYS" FOR PULL WITH REBASE BY DEFAULT

3 DO WORK IN YOUR FEATURE BRANCH, COMMITTING EARLY AND OFTEN

4 REBASE FREQUENTLY TO INCORPORATE UPSTREAM CHANGES

```
git fetch origin master
git rebase origin/master
```

REBASE AGAINST THE UPSTREAM FREQUENTLY TO PREVENT YOUR BRANCH FROM DIVERGING SIGNIFICANTLY.

ALTERNATIVE:

```
git checkout master
git pull
git checkout feature-1185-add-commenting
git merge master
```

THIS IS OFTEN DONE BY CHECKING MASTER OUT AND PULLING, BUT THIS METHOD REQUIRES EXTRA STEPS AS ABOVE

5 INTERACTIVE REBASE (SQUASH) YOUR COMMITS

WE WANT THE REBASE TO AFFECT ONLY THE COMMITS WE'VE MADE TO THIS BRANCH, NOT THE COMMITS THAT EXIST ON THE UPSTREAM. TO ENSURE THAT WE ONLY DEAL WITH THE "LOCAL" COMMITS

```
git rebase -i origin/master
```

GIT WILL DISPLAY AN EDITOR WINDOW WITH A LIST OF THE COMMITS TO BE MODIFIED

```
pick 3dcd585 Adding Comment model, migrations, spec
pick 9f5c362 Adding Comment controller, helper, spec
pick dcd4813 Adding Comment relationship with Post
```

NOW WE TELL GIT WHAT WE TO DO. CHANGE THESE LINES.

```
pick 3dcd585 Adding Comment model, migrations, spec
squash 9f5c362 Adding Comment controller, helper, spec
squash dcd4813 Adding Comment relationship with Post
```

SAVE AND CLOSE THE FILE. THIS WILL SQUASH THESE COMMITS TOGETHER INTO ONE COMMIT AND PRESENT US WITH A NEW EDITOR WINDOW WHERE WE CAN GIVE THE NEW COMMIT A MESSAGE.

6 MERGE YOUR CHANGES WITH MASTER

```
git checkout master
git merge feature-1185-add-commenting
```

MERGE YOUR CHANGES BACK INTO MASTER

7 PUSH YOUR CHANGES TO THE UPSTREAM

```
git push origin master
```

PUSH YOUR CHANGES TO THE UPSTREAM

BUG FIXES

SAME FLOW AS FEATURE DEVELOPMENT

1. PREFIX THE BRANCH NAME WITH “BUG” TO HELP YOU KEEP TRACK OF THEM
2. DO WORK IN YOUR BUGFIX BRANCH, COMMITTING EARLY AND OFTEN
3. REBASE FREQUENTLY AGAINST THE UPSTREAM
4. USE AN INTERACTIVE REBASE TO SQUASH ALL THE COMMITS TOGETHER

WITH A BUGFIX, SQUASH THE COMMITS DOWN INTO ONE AND EXACTLY ONE COMMIT THAT COMPLETELY REPRESENTS THAT BUGFIX. HALF OF A BUGFIX IS USELESS!

REFERENCES

"A SUCCESSFUL GIT BRANCHING MODEL" BY VINCENT DRIESSEN

HTTP://NVIE.COM/POSTS/A-SUCCESSFUL-GIT-BRANCHING-MODEL/

"A GIT WORKFLOW FOR AGILE TEAMS" BY REIN HENRICHS

HTTP://REINH.COM/BLOG/2009/03/02/A-GIT-WORKFLOW-FOR-AGILE-TEAMS.HTML

"MERGE OR REBASE" BY ATLASSIAN SOURCETREE

HTTP://BLOG.SOURCETREEAPP.COM/2012/08/21/MERGE-OR-REBASE/

"GIT PULL --REBASE BY DEFAULT" BY DEAN STRELAU

HTTP://D.STRELAU.NET/POST/47338904/GIT-PULL-REBASE-BY-DEFAULT

"A REBASE WORKFLOW FOR GIT" BY RANDY FAY

HTTP://WWW.RANDYFAY.COM/NODE/91

"A DEEP DIVE INTO THE MYSTERIES OF REVISION CONTROL" BY DAVID SORIA PARRA

HTTP://BLOG.EXPERIMENTALWORKS.NET/2009/03/MERGE-VS-REBASE-A-DEEP-DIVE-INTO-THE-MYSTERIES-OF-REVISION-CONTROL

BACKGROUND IMAGES

HTTP://MENGSHUEN.DEVIANTART.COM/ART/TREE-BRANCH-72007383

HTTP://WWW.PHOTOLIZER.COM/PHOTO%20STUDIES/MATERIAL/CONCRETE/3WALL_TEXTURE_BIG_101123.JPG

WE RECOMMEND GITIMMERSION.COM FOR LEARNING GIT BY PRACTICING

GIT IMMERSION

GIT IMMERSION IS A GUIDED TOUR THAT WALKS THROUGH THE FUNDAMENTALS OF GIT, INSPIRED BY THE PREMISE THAT TO KNOW A THING IS TO DO IT.

Git is a powerful, sophisticated system for distributed version control. Gaining an understanding of its features opens to developers a new and liberating approach to source code management. The surest path to mastering Git is to immerse oneself in its utilities and operations, to experience it first-hand.

Install Git

MAC WINDOWS

Graphical Clients

MAC WINDOWS

START GIT IMMERSION

WE PROVIDE PRIVATE TRAINING

New Context helps organizations access the full potential of the latest technologies and methodologies by offering structured training in areas such as Git, Ruby, Ruby on Rails, and Agile development. We specialize in building better software and staying on top of rapidly changing technologies. Let us bring that expertise to your business.

EMAIL:

LEMIORHAN@GMAIL.COM

TWITTER:

HTTP://WWW.TWITTER.COM/LEMIORHAN

LINKEDIN:

HTTP://WWW.LINKEDIN.COM/IN/LEMIORHAN

BLOG:

HTTP://WWW.FLYINGTOMOON.COM

LEMİ ORHAN ERGİN

LEMIORHAN.ERGIN@EU.SONY.COM

SONY

INFORMATION SYSTEMS EUROPE

SOLUTION DELIVERY ISTANBUL