ADVANCED SOURCE CONTROL: GIT

Nadeem Ansari
Sai Kiran Bandaru
Lalit Kandriga
Sandeep Velaga

CONTENTS

WHAT IS GITHUB FLOW?

A lightweight, branch-based workflow that supports teams and projects where deployments are made regularly.
Its simplicity gives it a number of advantages. But its main advantage is it's easy for people to understand, which means they can pick it up quickly and they rarely if ever mess it up

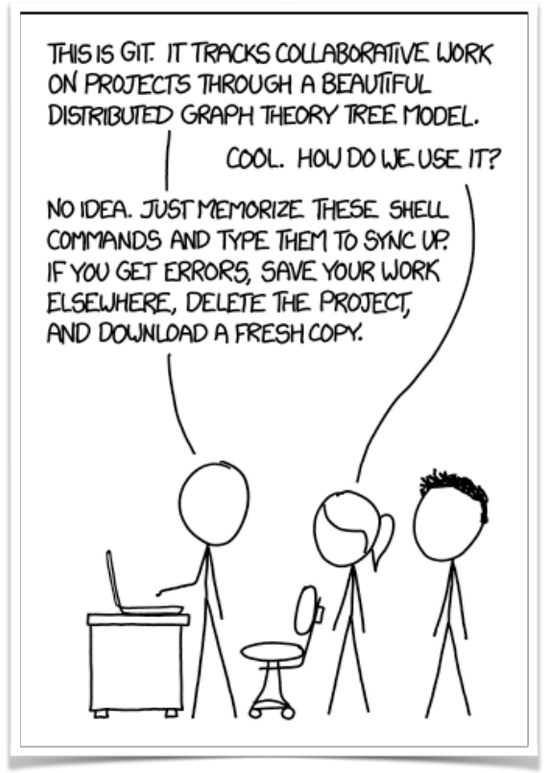


Fig.01: Git Control

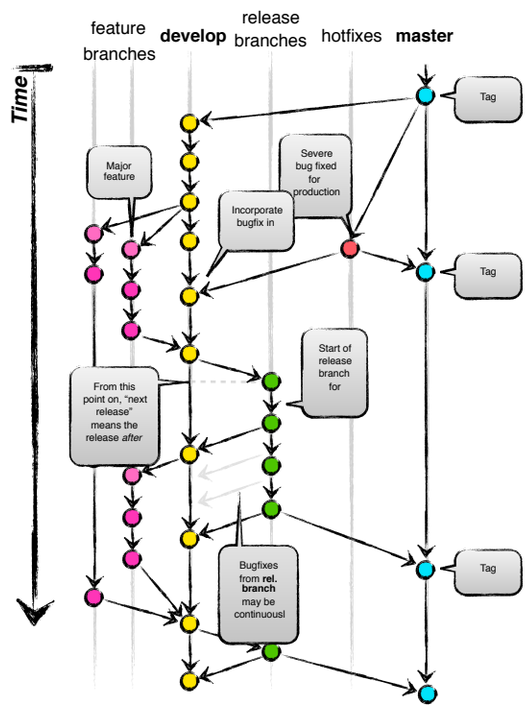


Fig.02.a: Git Flow

vs

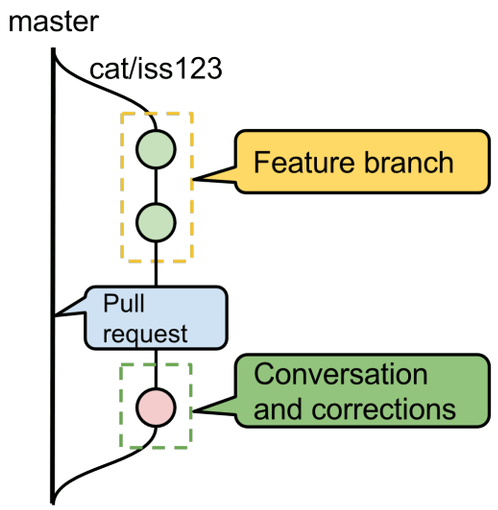
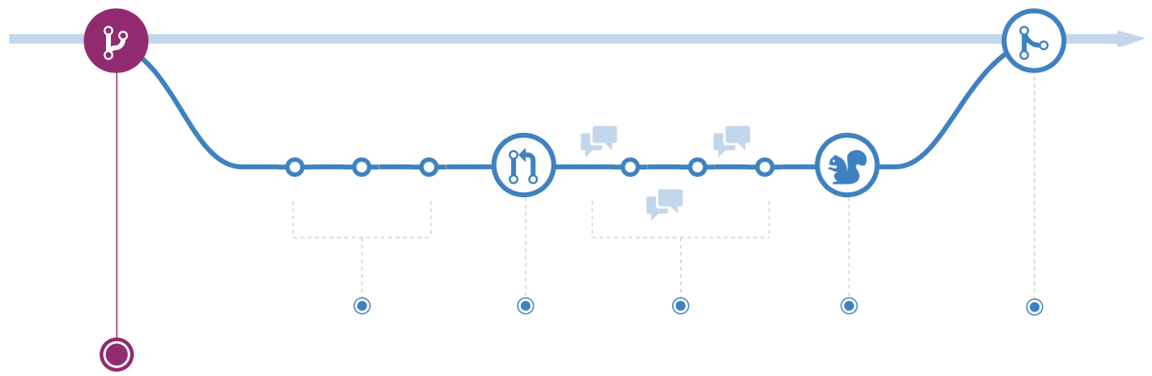


Fig.02.b: GitHub Flow



Branching

BRANCHING

- Core concept in Git
- Two branching strategies: Git Flow and GitHub Flow
- Basis for GitHub flow

Pt-2: Perhaps the most well-known branching strategy is Git Flow, which is a very comprehensive strategy. So comprehensive, in fact, it needs a whole set of scripts in order to use it properly! In my experience, Git Flow is too much for all but very large and technically advanced teams that are solving problems across multiple releases at one time.

BRANCHING

- “Anything in **master** is deployable”
- Branch name should be descriptive
- New feature completion is followed by merge request

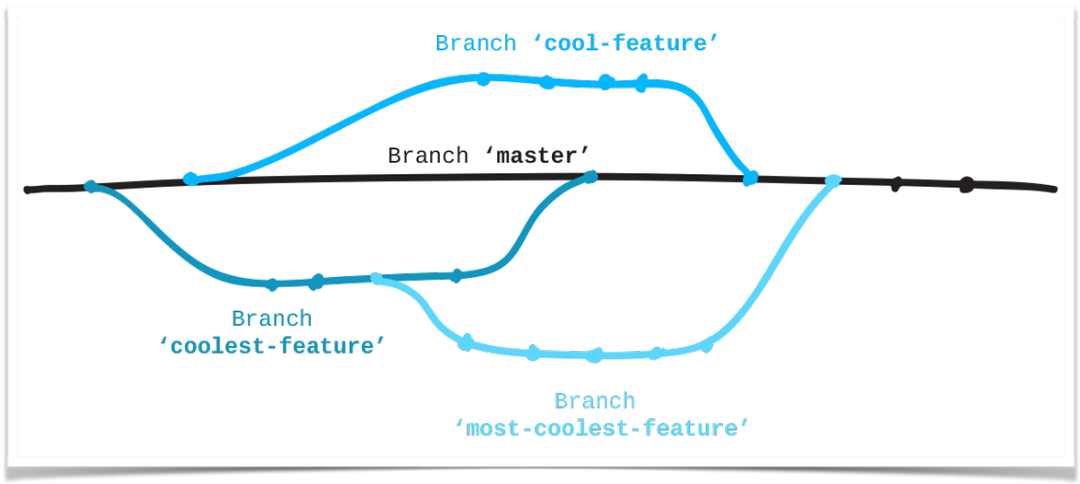
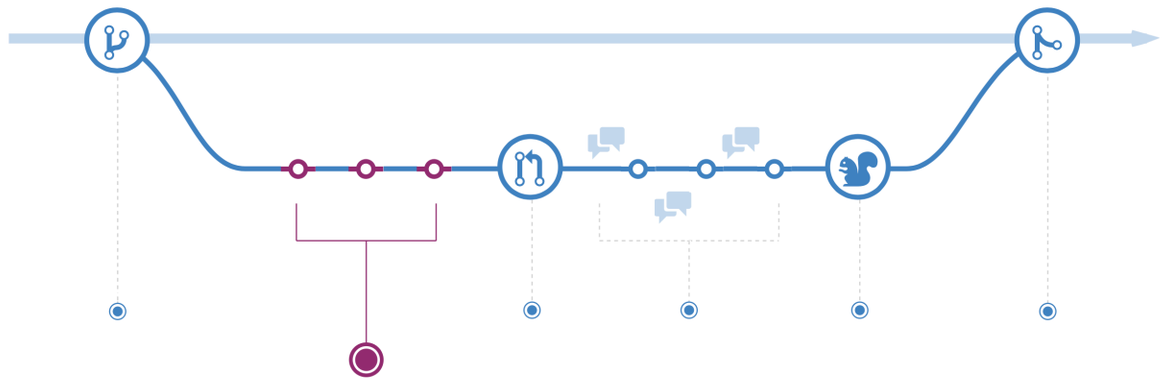


Fig.03: Branching in GitHub Flow

BRANCHING

- Changes in branch won't affect master
- Suitable for continuous deployment projects
- Made famous by Scott Chacon

Pt.2.
This is suited for projects that deploy continuously rather than around the concept of releases and so it's less constrained than git-flow.



Commit

COMMIT: SPLIT CHANGES

- Small means quicker & easier
- Easier to revert
- Easier to bisect
- Easier to browse

COMMIT: THINGS TO AVOID

- Mixing whitespace changes with functional code changes
- Mixing two unrelated functional changes
- Sending large new features in a single giant commit

COMMIT: THINGS TO AVOID

Never assume:

- Reviewer understands what the original problem was.
- Reviewer has access to external web services/site.
- Code is self-evident/self-documenting.

GOOD COMMIT MESSAGES

- Describe the change
- Should hint at code structure
- Sufficient information
- Limitations of code
- First line is important
- No patch-specific comments

REQUIRED REFERENCES

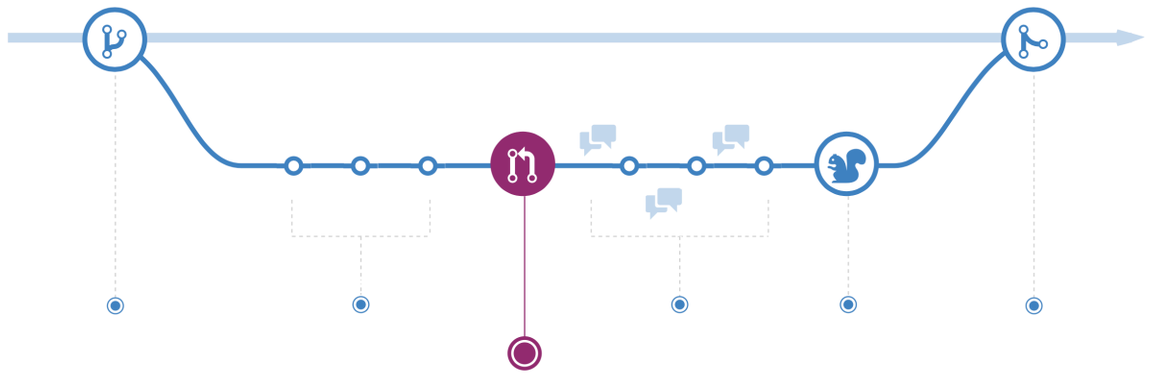
- Required References
  - Change-id
  - Bug
  - Blueprint
- Optional References
  - DocImpact
  - SecurityImpact
  - UpgradeImpact

```
Switch libvirt get_cpu_info method over to use config APIs

The get_cpu_info method in the libvirt driver currently uses
XPath queries to extract information from the capabilities
XML document. Switch this over to use the new config class
LibvirtConfigCaps. Also provide a test case to validate
the data being returned.

Closes-Bug: #1003373
Implements: blueprint libvirt-xml-cpu-model
Change-Id: I4946a16d27f712ae2adf8441ce78e6c0bb0bb657
```

Fig.04: A good commit message



Pull Request

PULL REQUEST

- Initiates discussion about commits
- Invoked any time during development
- Never force push

PULL REQUEST: WHAT CAN YOU DO?

- Open pull request
- Merging a pull request
- Close a pull request
- Delete unused branches

PULL REQUEST: WHEN TO DO?

- Any point during development
- Stuck and need help/advice
- Need someone to review your work

PULL REQUEST: WHY?

- Any point during development
- Stuck and need help/advice
- Need someone to review your work

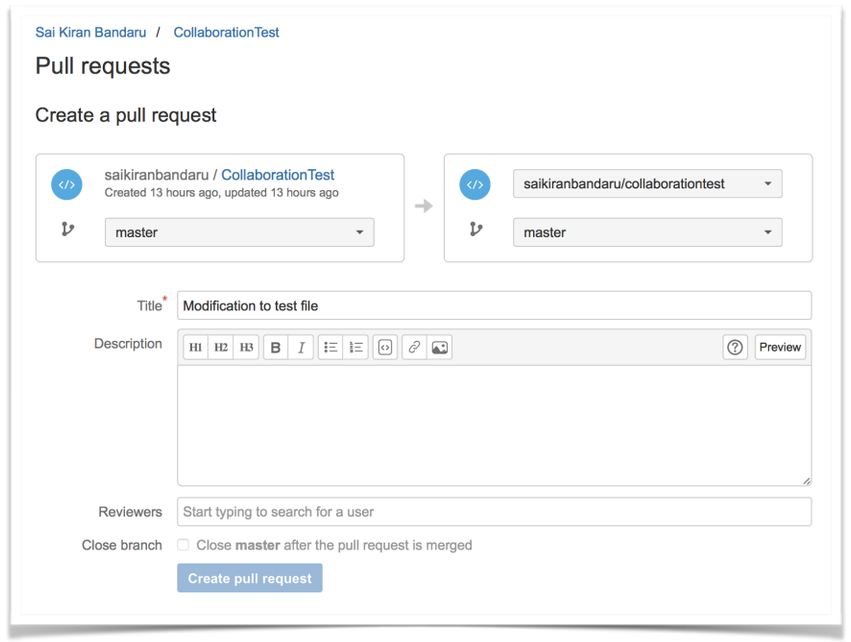
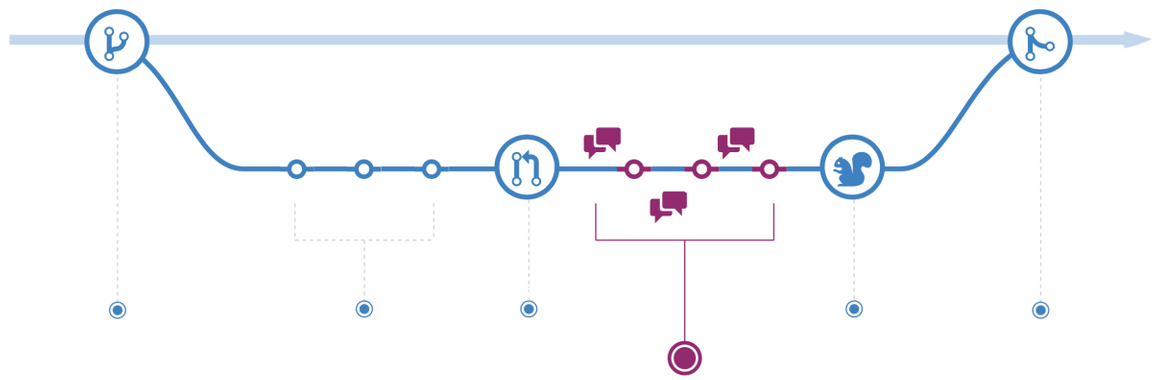


Fig.05: Pull request



Review

REVIEW

- Reviewing is recommended as a pair activity
- Only structure & syntax is reviewed, not functionality
- Host discussions to reach common understanding
- Introduce functional reviews for large features

- Reviewing is recommended as pair activity
- Only structure & syntax is reviewed not the functionality
- Discussions are hosted to reach common understanding
- Functional reviews for large and public facing features to be introduced

REVIEW: POST-COMPLETION

- Upon completion, GitHub issue is labeled as **reviewed**
- Feature is ready to be merged
    - Developer updates the pull request
    - Explains why he/she did otherwise

- When the review is done it either gets merged or, if the reviewer leaves comments, the GitHub issue is labeled as reviewed.
- The feature is ready to be merged as soon as the developer updates the pull request or explains why he/she did otherwise.

REVIEW: POST-COMPLETION

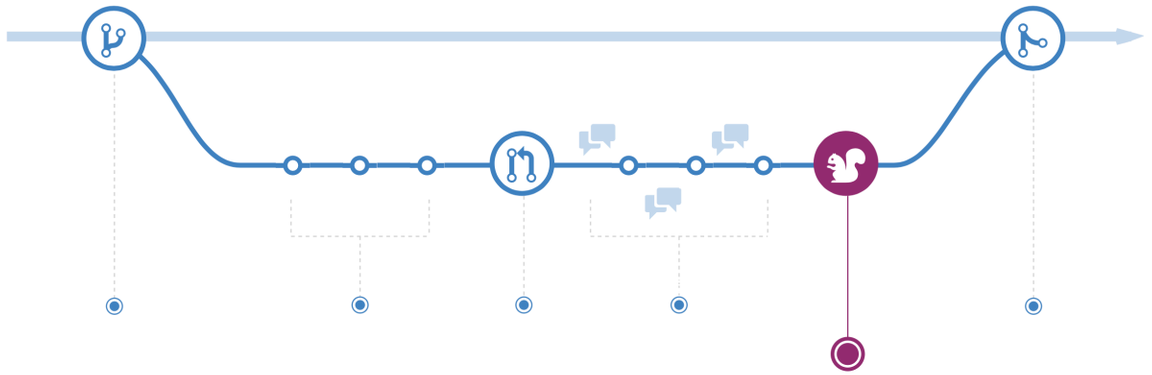
- Challenge to get teams to review quickly
- Asynchronous for smaller features
- Group gathering for larger features

- One recurring challenge is getting the teams to review as fast as possible without interrupting their work. Its asynchrounous for smaller features and group gathering for larger features.

REVIEW: TOOLS

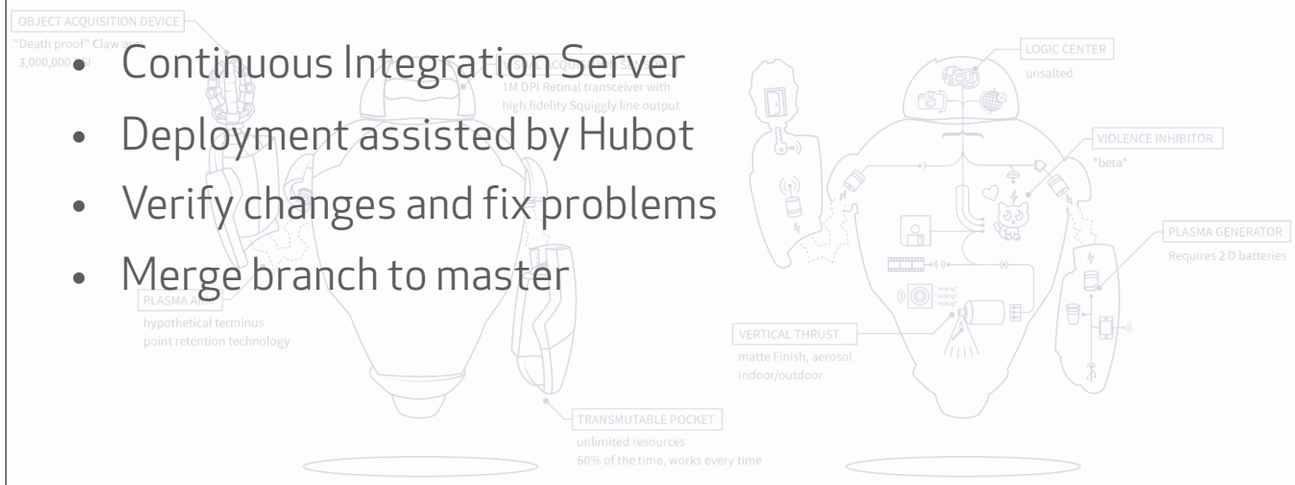
- Finding search objects faster
- Increase in flexible review
- Comment viewing with more context
- Resume from last work point

Pull requests with many changes need reviewing from diff people with diff expertise.
If you’re a Ruby expert, for example, you might want to focus just on the Ruby code and ignore any changes made to HTML and CSS files.
You can either search by extensions like .rb, .html, etc. or by filename if you know its name.
The most popular style on GitHub is reviewing all changes in a pull request at once, making the pull request the unit of change.
Some teams choose to use a commit-by-commit workflow where each commit is treated as the unit of change
There is a lot of conversations and additions that goes on during a code review.
These help new developer gain context & understand how and why a codebase has evolved over time.
With a project with many incoming pull requests it is not possible to do a code review in one day. It is convenient to view just the changes that have occurred after reviewing the last pull request



Deployment

DEPLOYMENT



- Continuous Integration Server
- Deployment assisted by Hubot
- Verify changes and fix problems
- Merge branch to master

Changes made are pushed to a branch. The build is then to pass on the Continuous Integration Server. Hubot helps in deploying it. Verify if the changes work and fix any problems that arise. Merge the branch back into master.

Continuous Integration Server - (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

Hubot - Hubot is an automation tool that can sync with other chat services.

Janky is a continuous integration server built on top of Jenkins, controlled by Hubot, and designed for GitHub.

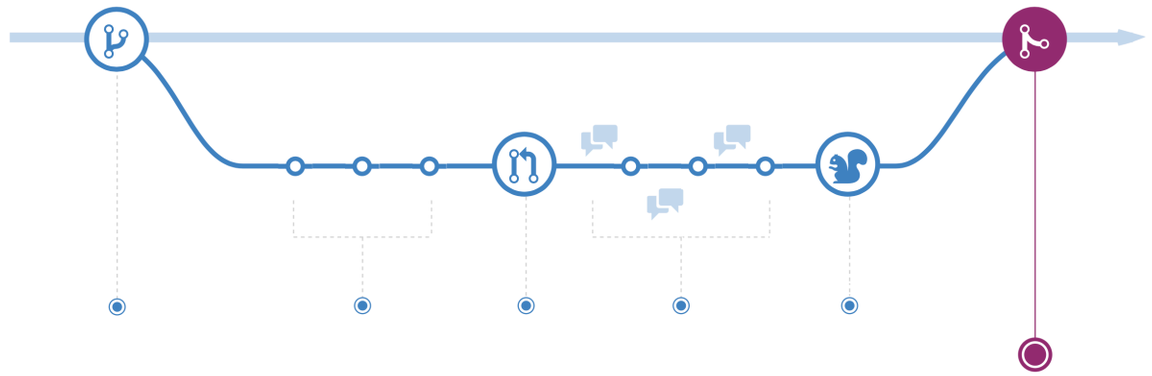
SAFETY MEASURES

- Pre-deployment, Janky's CI build status should be green
- Deployment is locked during use
- Check performed to ensure recent commit message

SAFETY MEASURES

- Changes to be observed for 15 min
- Merging happens through pull requests
- Post-merge, the deployment area re-opens

Small to moderate changes to be observed running correctly for 15 minutes
If everything is good merging happens through pull requests.
After the branch gets merged into master the deployment area is open again.



Merging

MERGING

- After verification, code is merged into master
- Test merge locally before merging into GitHub repo
- New work will be based off of merged branch to master

After changes being verified in production, the code is merged into master branch
It maybe wise to test this merge locally before merging into GitHub repository.
Any new work will be based off of this merged branch to MASTER.

MERGING

- Post-merge, pull requests store all changes made
- Helps people revisit their code
- Understand the decisions taken

After merge, the pull requests store all the changes made to the code
This helps people to revisit their code and understand how/why a particular decision was taken.

DEMONSTRATION OF GITHUB FLOW