Git for Teams of One or More

Emma Jane Westby

Twitter: emmajanehw

slides: https://github.com/emmajane/gitforteams

Hello! My Name is Emma

I have been using version control for 10+ years and had the great misfortune of teaching CVS to arts majors before distributed version control was a thing.

Warning!

This is not a talk about all the commands you can run in Git.

Resources for Commands:

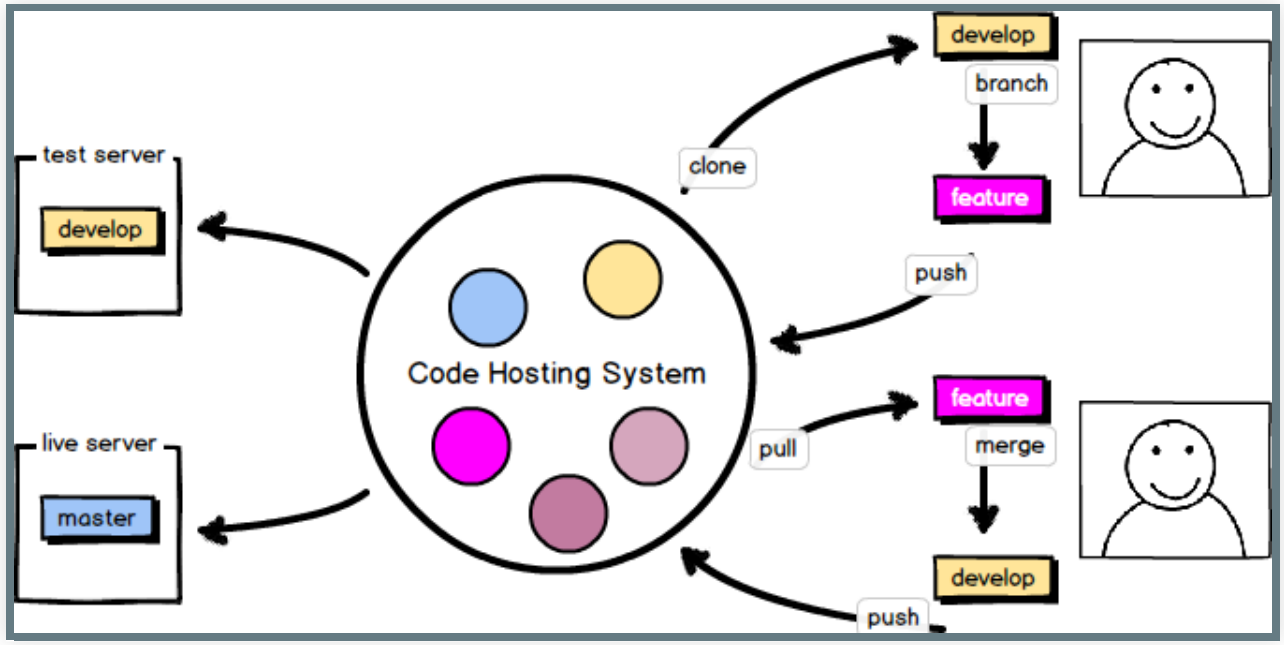
- Mega Resources List o' Links
- Git Documentation
- Pro Git

My Goal for this Workshop

By the end of this session you should be able to:

- Determine a permission strategy for your project.
- Determine a branching strategy for your project.
- Create documentation which outlines how your team members will use version control.

Workshop Outcome: Personalized Documentation



---

this is where we want to end up by the end of today. You know where each branch lives. You know how / where a branch is closed.

github.com/emmajane/gitforteams

You'll want a copy of the slides for reference as we go through the activities. Please open this page now.

Warm-up Exercise

People and Process Before Commands and Code

Sample (Rhetorical?) Questions

- Who has commit access?
- Why do you know your code isn't broken?
- Does your team use test-driven development?
- Do you have an idependent quality assurance team?
- Can you deploy "broken" code?

Activity: Identify Current R&R

1. Write down a list of all of the people/roles on your code team.
2. Write a list of the tasks these people/roles are responsible for code-wise.

R&R = roles and responsibilities

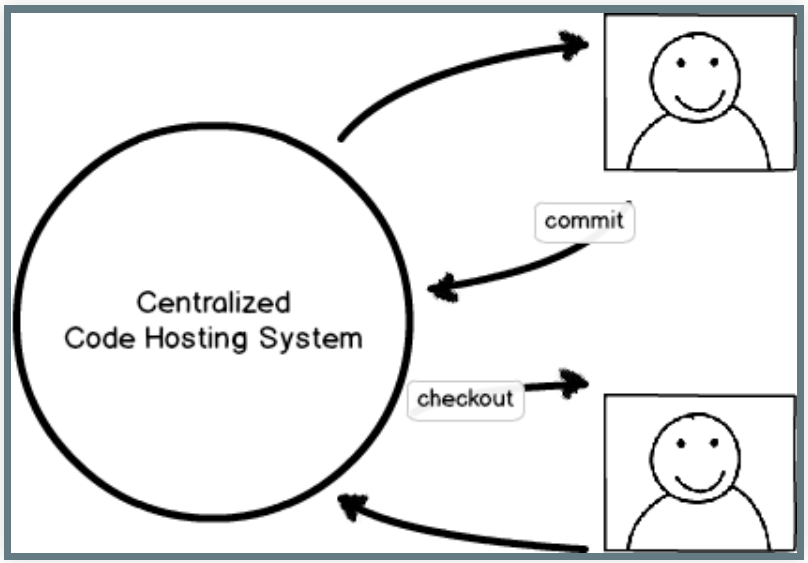
Activity: Sketch the Assembly Line

Sketch a timeline of how you'd like new code to be incorporated into your project. Is there a review processs? A test suite? Minimal barriers to code commits?

(You will to refine this sketch during the workshop. There are no wrong answers right now.)

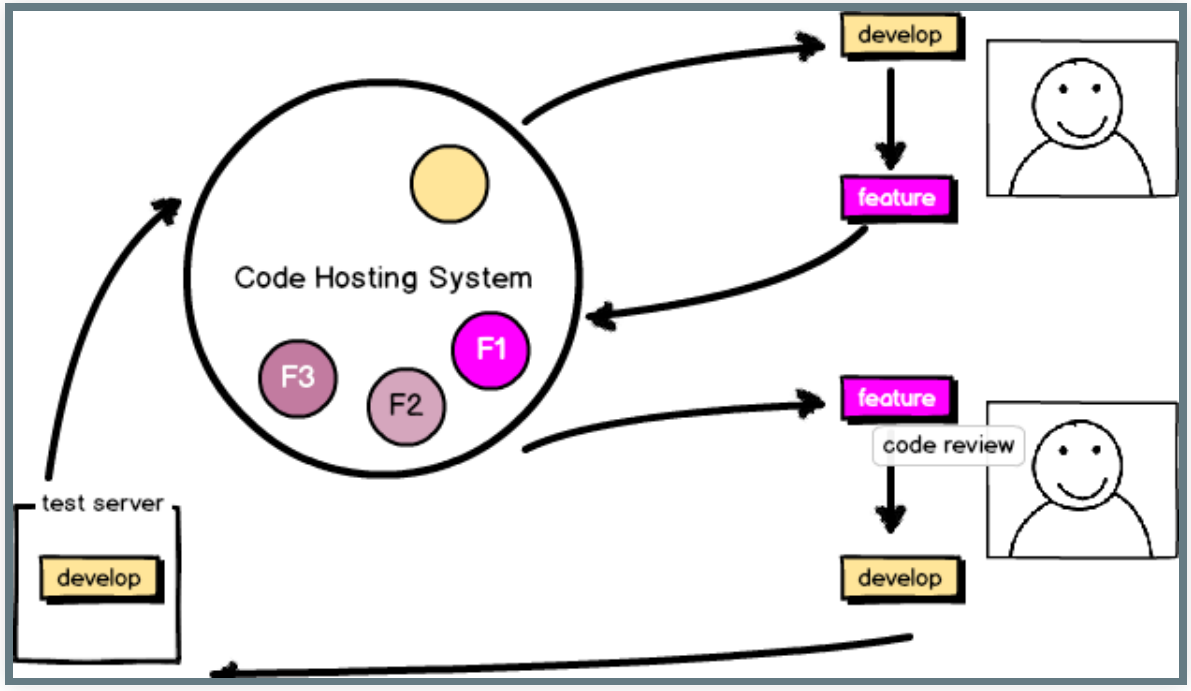
Sample Activity Answer: Centralized

Everyone works in the same centralized repository. There's no peer review or testing.



Sample Activity Answer: Pre-Merge QA Team

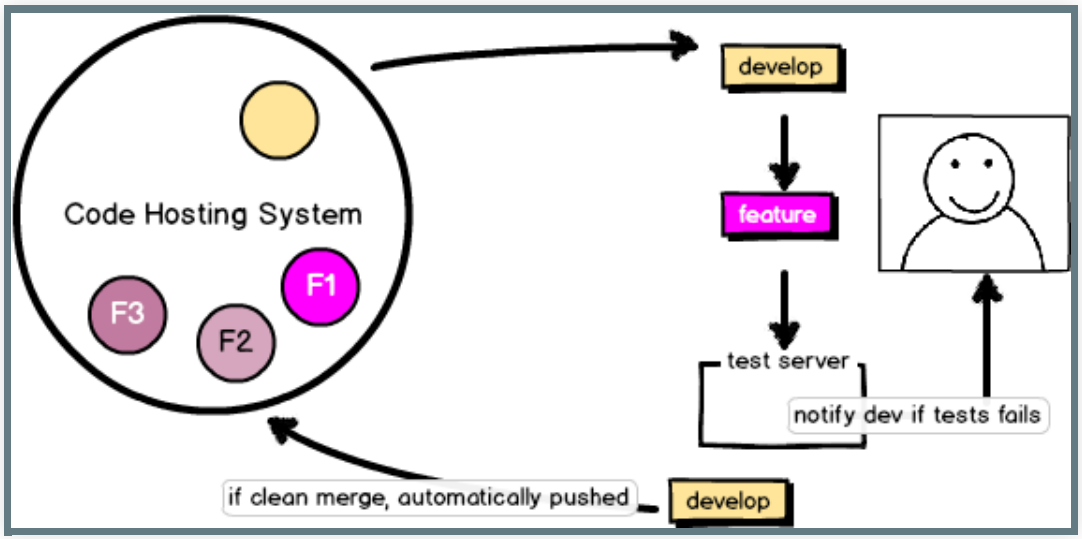
A quality assurance team, and optional test suite, decide if your work is acceptable.



or this could be just an automated testbot

Sample Activity Answer: CI or Post-Merge Test Suite

A testbot notifies you if your work is not acceptable (possibly after adding it to the main branch).



CI assumes everything is good, but notifies you if it's not.

Part 1

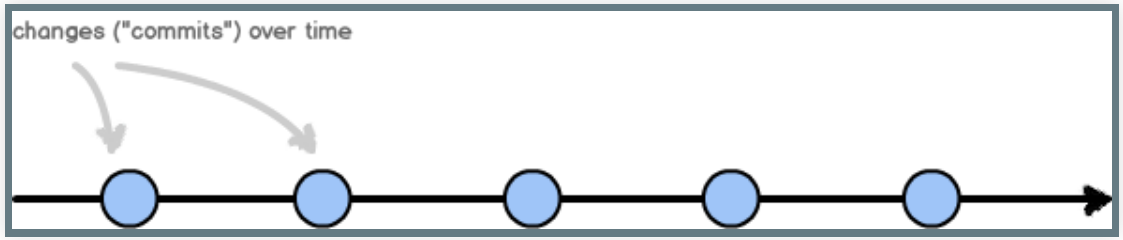
Project Hosting

When you first create a Git project, you will need to decide who can commit their code to the repository.

Step 1: Identify and describe the governance for your code.

Centralized: Trust Everyone

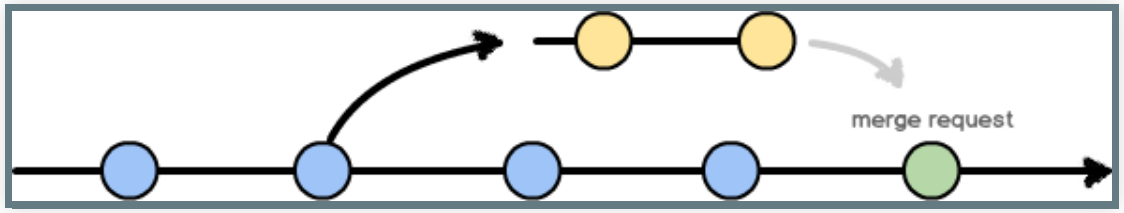
Everyone has read-write access to the same repository on a centralized disk (e.g. subversion). This is also how you work *locally* with Git.



- Pro: Author has to deal with their own merge conflicts.
- Con: No guarantee the code works.

Forked: Trust No One

Project forks give full permissions to developers so they can do work. New work is added to the main project through a request to the upstream project.

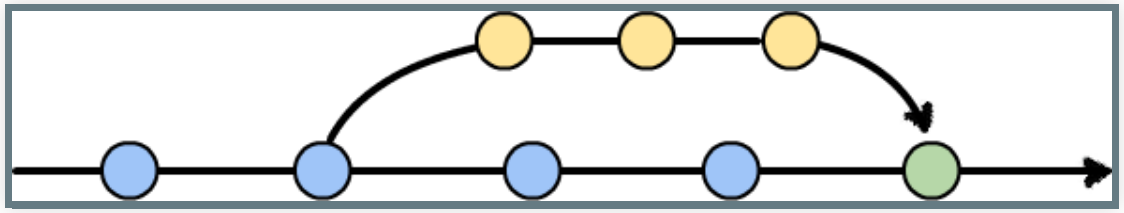


- Pro: Forces a review process.
- Pro: Encourages experimentation.
- Con: Private repos must be duplicated per team member.
- Con: More steps to incorporate new work.

This is the default strategy for public code repositories with open access for viewing the project.

Branched: Trust the Process

Developers work in a branch of the centralized code repository. Only the politics of the project prevent them from committing their work to the main body of work.



- Pro: Encourages clean/working master.
- Con: Must give explicit write permission to all team members.
- Pro/Con: Encourages, but does not **require** code review.

This is the default strategy for private code repositories with named team members.

Your Home Work

- If you choose **BRANCHED**, you need to setup a PRIVATE repository for your code, and grant permission to all team members to push their changes to the server.
- If you choose **FORKED**, you need to setup PUBLIC or PRIVATE repository for your code, and ensure all team members to can create their own PUBLIC or PRIVATE copy of the project, AND submit merge requests to the main project.

Part 2

Separating Collated Code with Branching Strategies

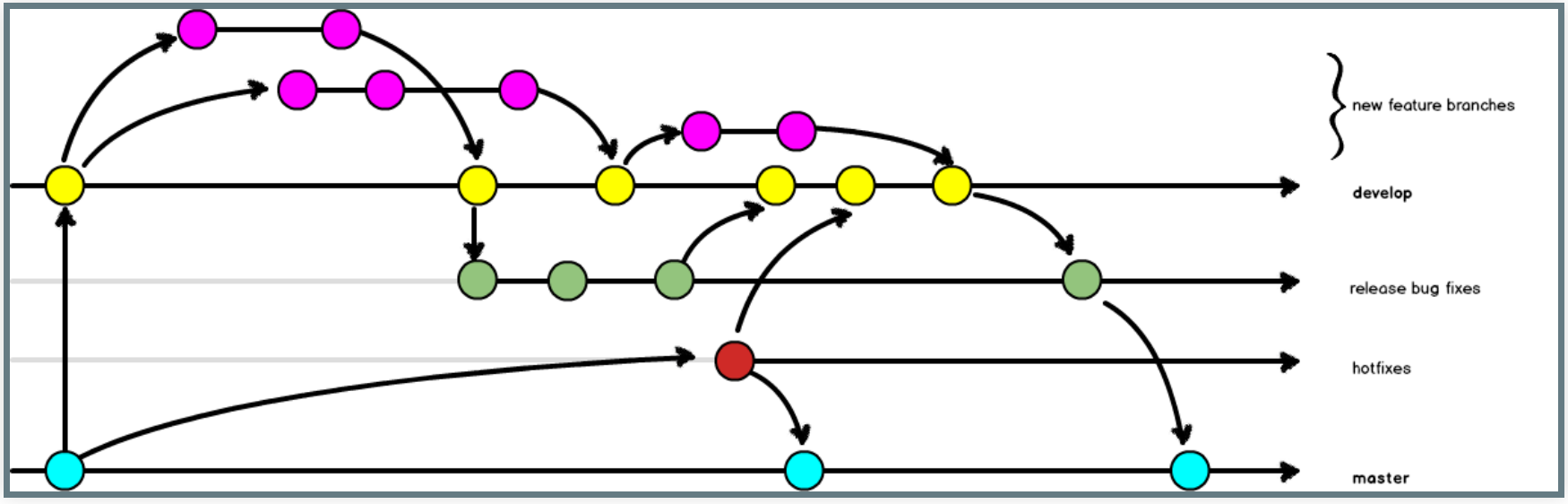
Identify and describe how your code is collated within your repository.

Branching Strategies

- Scheduled Release: Gitflow or Simplified Gitflow
- Continuous Deployment: Branch Per Feature or GitHub Flow

Scheduled Release

- Optimized for the collation of many smaller changes into a single release.
- Typically used for a downloadable product; or web site with a scheduled release cycle (e.g. "Wednesdays").
- Incorporates human-reviews, and possibly automated tests.

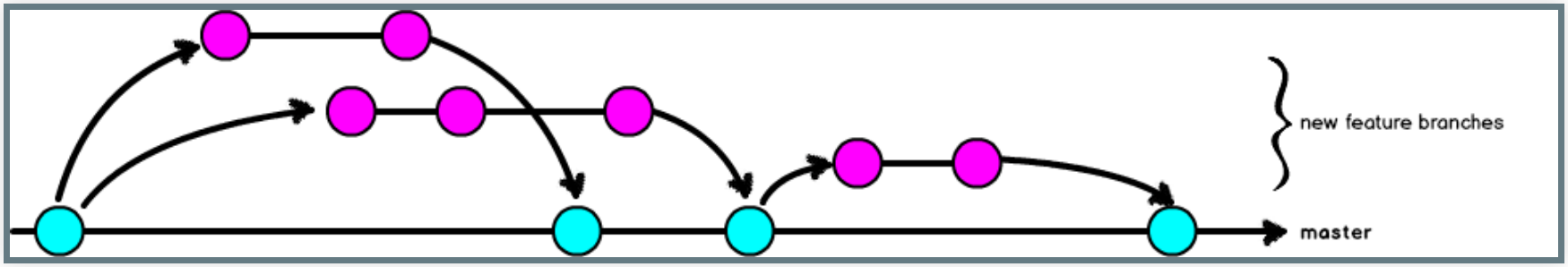


---

if you have the concept of stable releases, hotfixes, point releases, security releases, multiple supported versions, etc, then you need this granularity for your branches. There is always a period of time where you do not trust your

Continuous Deployment

- Code is deployed faster than scheduled releases.
- Requires (trusted) test coverage.
- Typically uses a mechanical gatekeeper to check in code to the master branch.
- Often has flippers/flags for fine grained access to in-progress features.
- Fewer branches to maintain / keep updated.



---

if you don't need the granularity of mulitple supported versions, you can probably get away with something closer to this branching strategy. Can you get away with just tags? Do you intend to go back and work on a previous version? As soon as you have the concept of a separate security hotfix, you need to introduce a separate branch. In CD: everything is urgent, so there's not a separation of a really urgent security fix. (a deployed system)

Activity

Which best describes your current setup?

- Scheduled Release: Gitflow or Simplified Gitflow
- Continuous Deployment: Branch Per Feature or GitHub Flow

On the sketch diagram you created previously, add a CIRCLE (or a triangle, or a pony) around the collation points for code. These represent new branches. Where possible, REDUCE the number of collation points because merging out-of-date branches is a potential pain point.

Your Home Work

- If you choose **SCHEDULED RELEASE**, determine the points where code needs to be collated for release.
- If you choose **CONTINUOUS DEPLOYMENT**, codify how trust is deployed in your code.

Part 3

Commit Granularity

The Great Rebase Debate

What is a Commit

Record changes to the repository

How can we use Commits

- log
- gitk
- blame
- bisect

Sharing Work: A brief history lesson

The patch workflow and **`git am`**.

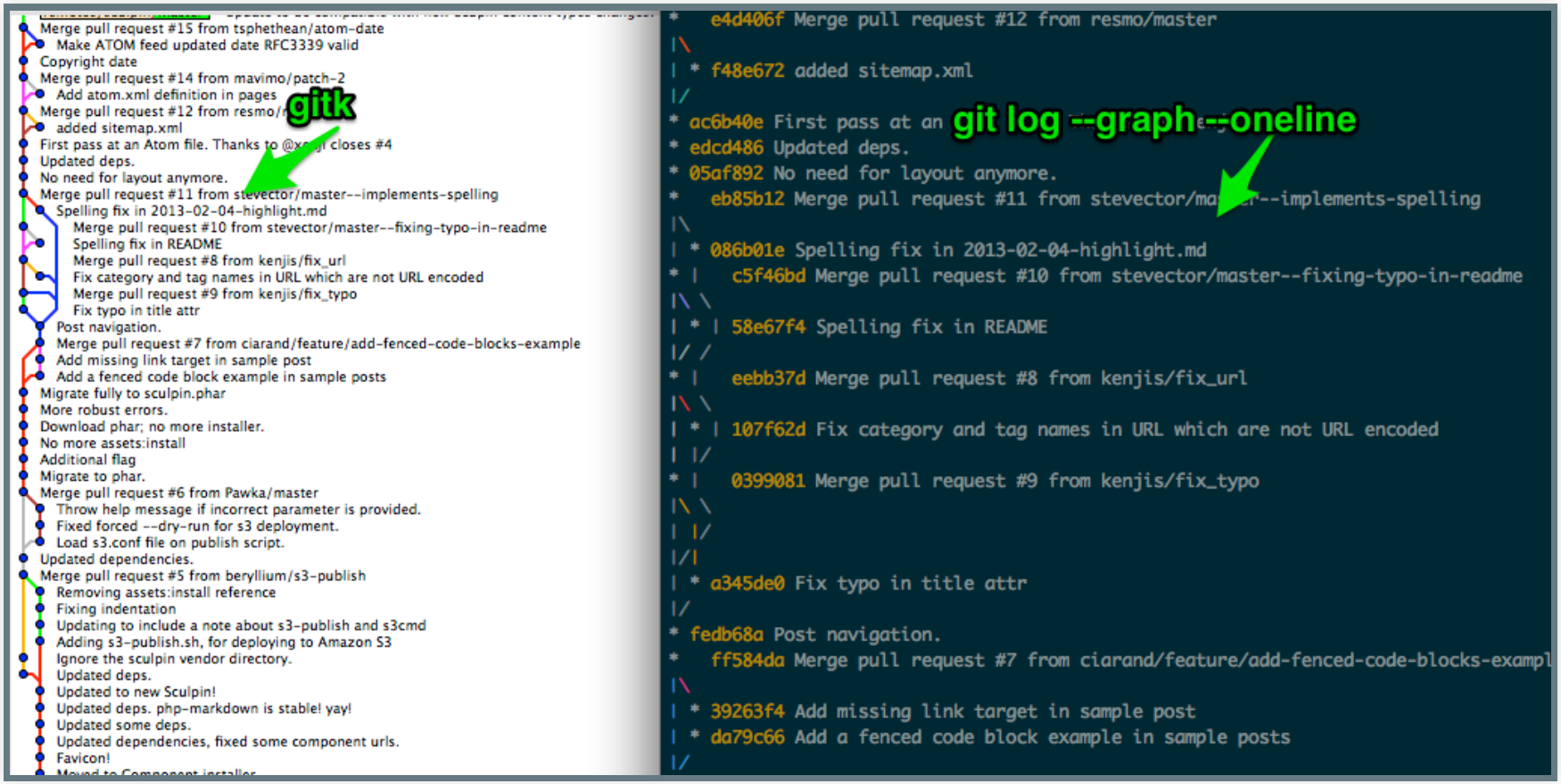
> *The commit message is formed by the title taken from the "Subject: ", a blank line and the body of the message up to where the patch begins.*

In other words: a commit is a whole idea.

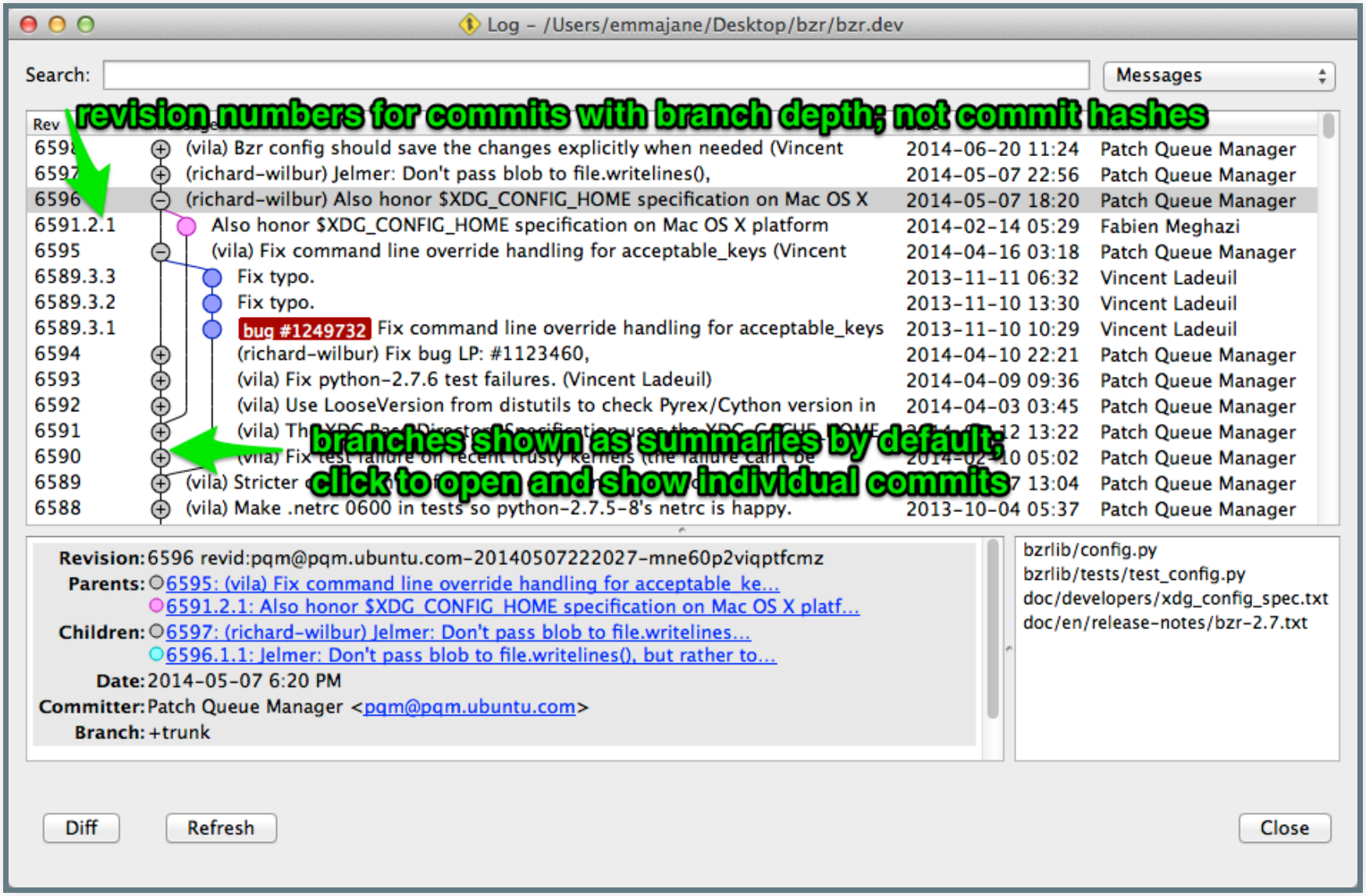
Sharing Work: Today

`git push`

Shares an entire branch, with all your micro commits.



Compare: bzr


Log – /Users/emmajane/Desktop/bzr/bzr.dev
Search:
Messages
revision numbers for commits with branch depth; not commit hashes
Rev
6596 (richard-wilbur) Also honor $XDG_CONFIG_HOME specification on Mac OS X 2014-05-07 18:20 Patch Queue Manager
6591.2.1 Also honor $XDG_CONFIG_HOME specification on Mac OS X platform 2014-02-14 05:29 Fabien Meghazi
6595 (vila) Fix command line override handling for acceptable_keys (Vincent 2014-04-16 03:18 Patch Queue Manager
6589.3.3 Fix typo. 2013-11-11 06:32 Vincent Ladeuil
6589.3.2 Fix typo. 2013-11-10 13:30 Vincent Ladeuil
6589.3.1 bug #1249732 Fix command line override handling for acceptable_keys 2013-11-10 10:29 Vincent Ladeuil
6594 (richard-wilbur) Fix bug LP: #1123460, 2014-04-10 22:21 Patch Queue Manager
6593 (vila) Fix python-2.7.6 test failures. (Vincent Ladeuil) 2014-04-09 09:36 Patch Queue Manager
6592 (vila) Use LooseVersion from distutils to check Pyrex/Cython version in 2014-04-03 03:45 Patch Queue Manager
6588 (vila) Make .netrc 0600 in tests so python-2.7.5-8's netrc is happy. 2013-10-04 05:37 Patch Queue Manager
branches shown as summaries by default;
click to open and show individual commits
Revision: 6596 revid:pqm@pqm.ubuntu.com-20140507222027-mne60p2viqptfcmz
Parents: 6595: (vila) Fix command line override handling for acceptable_ke...
6591.2.1: Also honor $XDG_CONFIG_HOME specification on Mac OS X platf...
Children: 6597: (richard-wilbur) Jelmer: Don't pass blob to file.writelines...
6596.1.1: Jelmer: Don't pass blob to file.writelines(), but rather to...
Date: 2014-05-07 6:20 PM
Committer: Patch Queue Manager <pqm@pqm.ubuntu.com>
Branch: +trunk
bzrlib/config.py
bzrlib/tests/test_config.py
doc/developers/xdg_config_spec.txt
doc/en/release-notes/bzr-2.7.txt
Diff
Refresh
Close

Problem!

Git tools are COMMIT-aware, not BRANCH-aware.

- gitk
- bisect

Solution!

`git rebase`

Forward-port local commits to the updated upstream head

In English: re-draw the graph for the commit history as if the rebased commits were already in the history when you did your work.

Solution!

```
git rebase -i
```

> *Make a list of the commits which are about to be rebased. Let the user edit that list before rebasing. This mode can also be used to split commits (see SPLITTING COMMITS below).*

In English: combine, or separate, any commits previously made.

Yes, Re-write History

Because the tools used to interpret history are crude, the recommended approach is simply to fix history.

TWITCH

But this is how Git works. So there you go.

Activity

???

- Do a rebase together.
- Discuss team dynamics and which tools you'll want to use (bisect, gitk, blame, etc)

Part 4

Putting it all Together

- These examples are pulled from Drupalize.Me when I was working as their PM and sometimes front end dev.
- This is a product with no external stakeholders.
- YMMV, YOLO, etc.

---

these are both in the resources for the repository

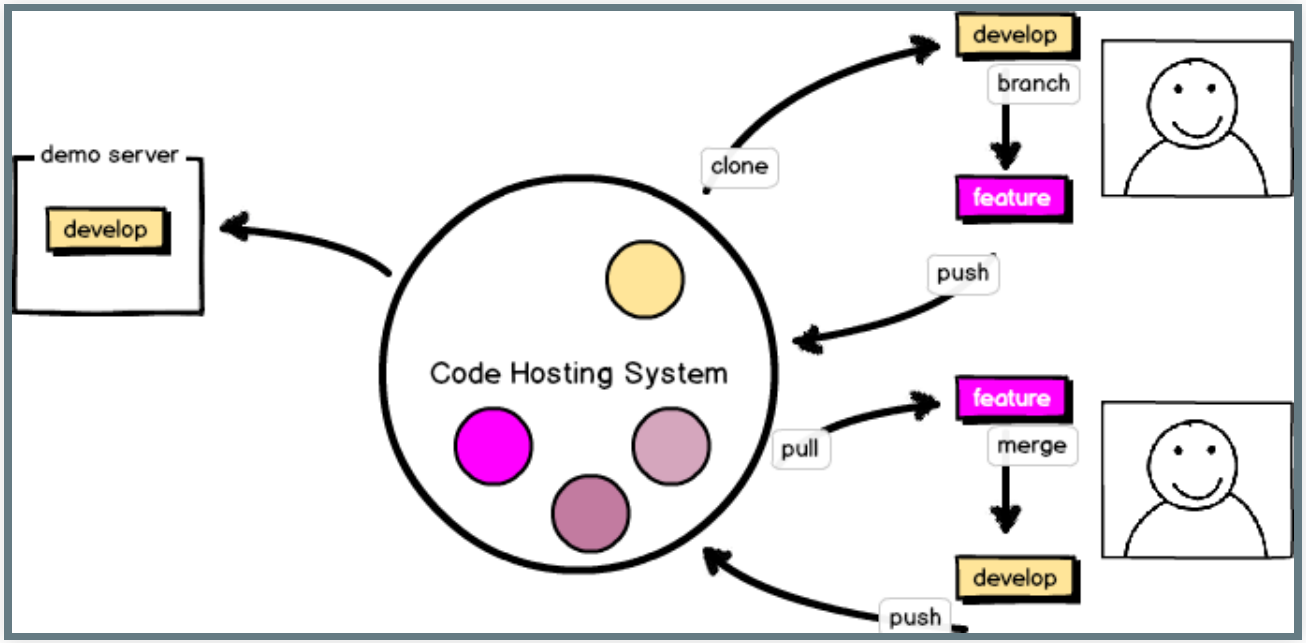
Project Highlights

- Drupal 6 -> Drupal 7 upgrade
- Aiming for speed of work, not stability.
- Changes were **not** being deployed to the live server.
- No weekly demos (which you might have for client work).
- Total time: 18 months.
- Star Wars Sprintflow

Some Notes on Naming

- Use terms which resonate with your team (MVP -> LBB).
- Giving a descriptive name to projects and processes allows you to change the meaning by changing the name.
- There are a lot of Ewoks.
- There are more My Little Ponies.

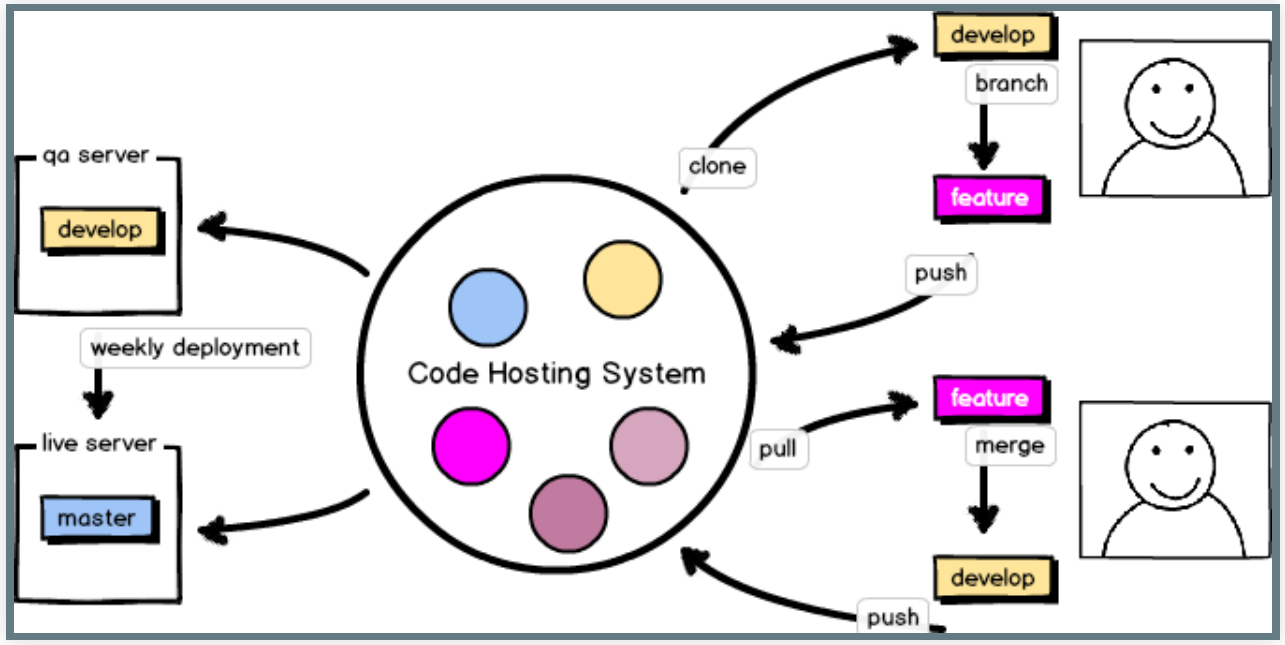
The Star Wars Workflow



---

pre-launch: peer review with branched permission strategy; separate QA server where work is available for review, but typically devs just look at their local version of the current dev branch.

Whispering Pines Workflow



- Aiming for stability first, speed second.
- Some test coverage.
- Changes are collated weekly onto a QA server, and deployed from there.

Documentation: - Whispering Pines Weekly Workflow - Release philosophy - Deployment

Final Activity: Sketch Your Workflow

- Restructure your previous diagrams to include the infrasture where code is collated.
- Add arrows to represent the direction code travels.
- To the arrows, add the git commands which you'd use.
- Create a written narrative which describes the EXACT commands people should use to move code through the process. (See previous slide for examples.)

Resources

- Developer Workflow
- Scheduled Release: Gitflow (Cheatsheet) or Simplified Gitflow
- Continuous Deployment: Branch Per Feature or GitHub Flow

Thanks!

Feedback Welcome!

@emmajanehw

https://github.com/emmajane/gitforteams