

Git for Teams of One or More

Emma Jane Westby

Twitter: [emmajanehw](https://twitter.com/emmajanehw)
[<http://twitter.com/emmajanehw>]

slides: <https://github.com/emmajane/gitforteams>
[<https://github.com/emmajane/gitforteams>]

Hello! My Name is Emma
[http://en.wikipedia.org/wiki/Emma_Jane_Hogbin]

@emmajanehw

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. This workshop is being turned into an O'Reilly book. Yay!

Warning!

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

Resources for Commands:

- Mega Resources List o' Links
[<http://developerworkflow.com/resources/offsite.html>]
- Git Documentation
[<http://git-scm.com/doc>]
- Pro Git
[<http://git-scm.com/book>]

Yes, the slides are uploaded

github.com/emmajane/gitforteams

On an index card...

Write down your answer to:

- REQUIRED: What (workflow-related) **questions** do you need answered today?
- OPTIONAL: How does Emma **get in touch** with you after the workshop to make sure your question(s) were answered?

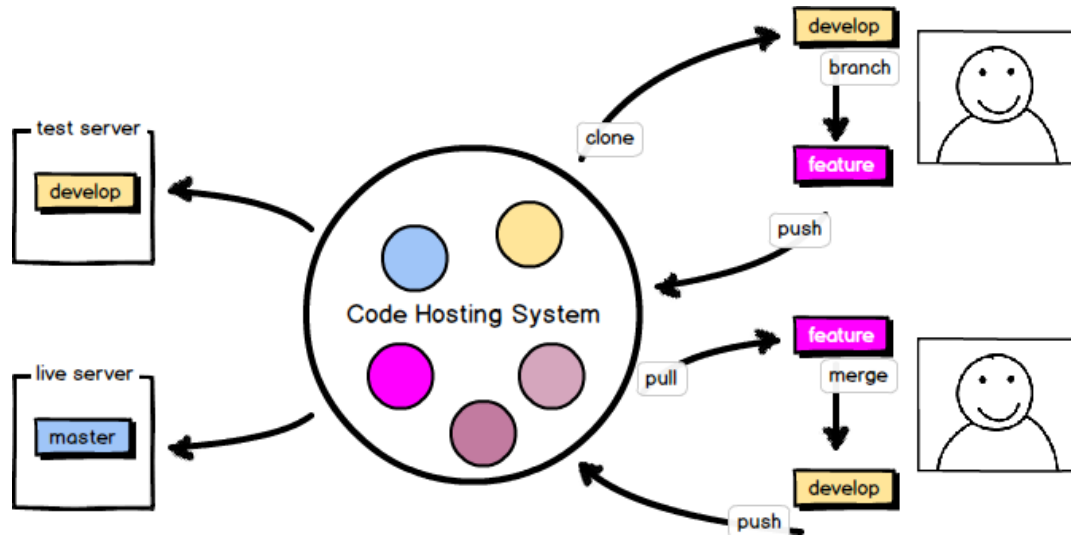
during the break I'm going to review the index cards to make sure we're on track.

My Goal for this Workshop

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

- Choose a **permission strategy** for your project.
- Choose a **branching strategy** for your project.
- Create project-specific **documentation** which outlines how your team members interact with your code.

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.

slides:
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**

Basic 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 independent quality assurance team?
- Can you deploy "broken" code?

we'll start with the easy questions you MUST be able to answer. Your current answers may help you to uncover problems with your current setup.

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

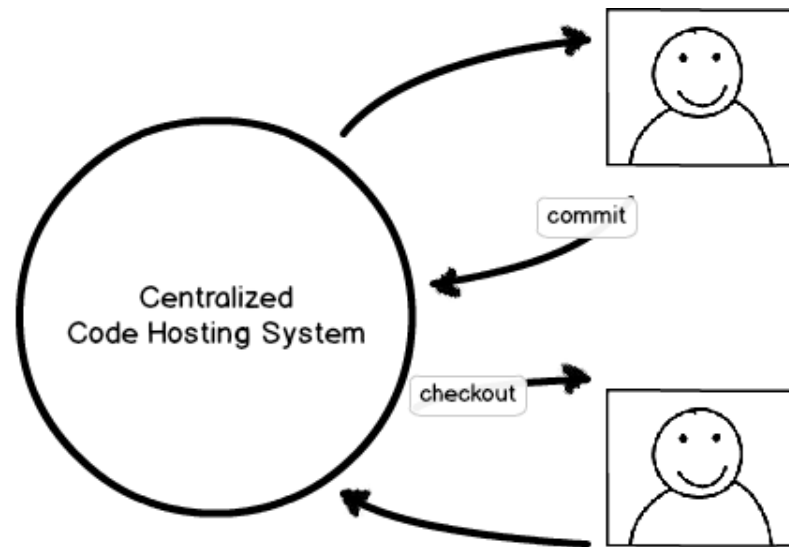
Activity: Sketch the Assembly Line

Sketch a time line of how code is incorporated into your project.

- Where do people grab the code from?
- How do people share their work? (branch? patch? fork?)
- Is there a review process?
- Are there barriers to code commits (test suite, QA team)?

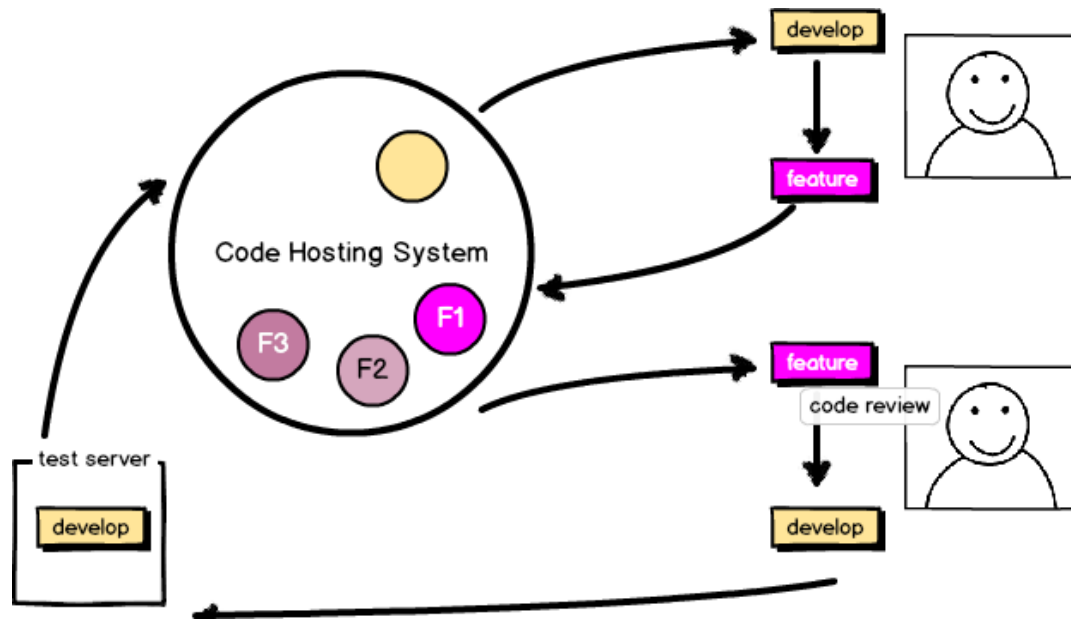
right now, there are no wrong answers. It's just a sketch. A rough approximation is fine.

Example: Centralized



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

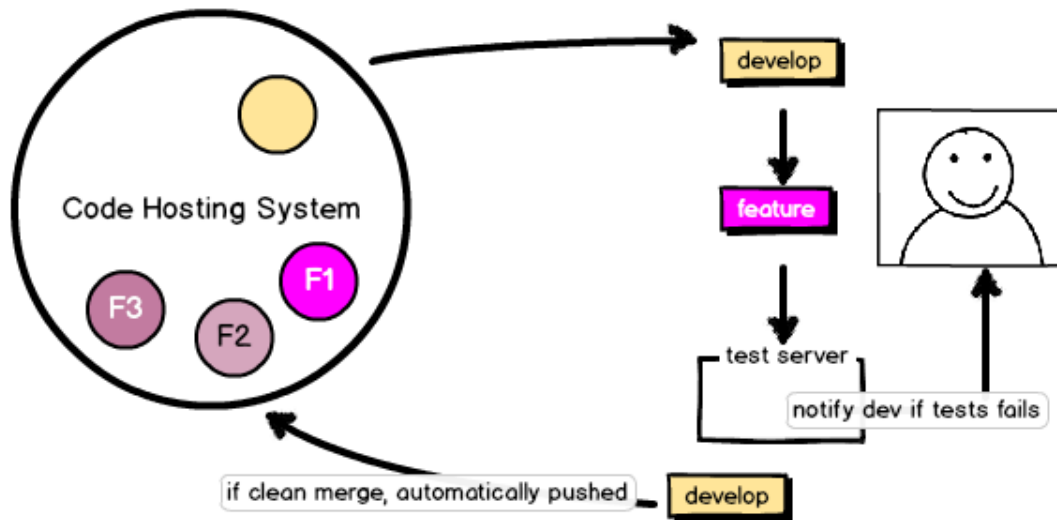
Example: Pre-Merge QA Team



or this could be just an automated testbot.

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

Example: 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.

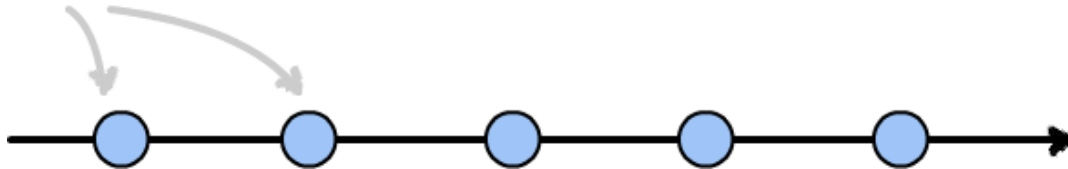
Overview

- **Centralized** - Trust Everyone
- **Patched** - Trust No One; Propose *a Solution*
- **Forked** - Trust No One; Show *Your Work*
- **Branched** - Trust the Process

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.

changes ("commits") over time



Centralized

Pro

Author has to deal with their own merge conflicts.

Con

No guarantee the code works.

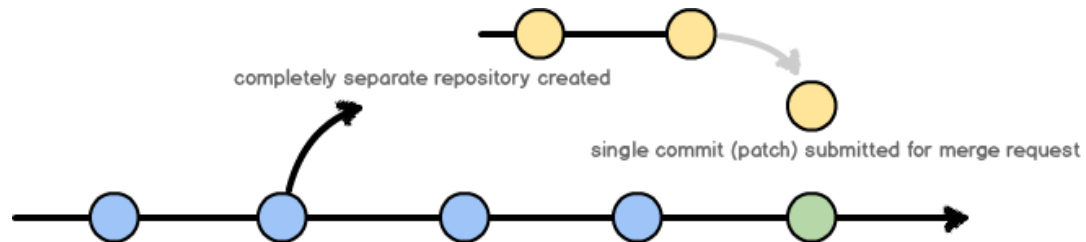
Examples of Centralized Workflow?

Projects using subversion, CVS, or other centralized systems.

Internal projects using a centralized system (e.g. subversion).

Patched: Trust No One; Propose a Solution

Everyone has read access. Very few have write access. Suggested changes are presented as whole ideas in a single patch file for review.



Patched: Trust No One; Propose *a Solution*

Pro

- Forces a review process.
- Works well with git tools (bisect, gitk).

Con

- Sharing work is more complicated than branching.
- Contributors (potentially) need to setup their own code hosting platform.

This is what git was optimized for. It's archaic and doesn't work well with web-based code hosting and ticketing platforms such as GitHub.

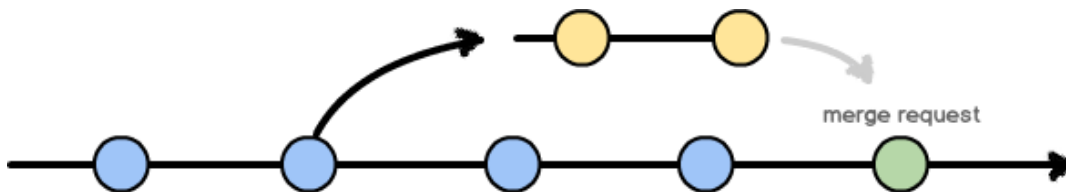
Examples of Patched Workflow?

- Linux
- Drupal
- FOSS projects still using a centralized code hosting model OR mailing-list code sharing model

Linux, Drupal

Forked: Trust No One; *Show Your Work*

Project forks give full permissions to developers so they can do work in any commit granularity they choose. New work is added to the main project through a request to the upstream project via a proposed **branch of commits**.



Forked: Trust No One; Show *Your Work*

Pro

- Forces a review process.

Con

- Commit granularity may prevent effective debugging.
- Private repos must be duplicated per team member.
- More steps to incorporate new work.

This is the default strategy for public code repositories with open access for viewing the project. Wrote a resource on why this may be bad at

<http://developerworkflow.com/resources/evolution-social-coding.html>

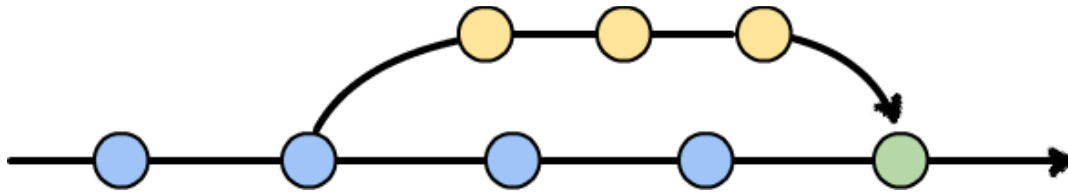
Examples of Forked Workflow?

- Django
- Ruby on Rails
- CakePHP
- FOSS projects hosted on GitHub

Django, Rails, CakePHP

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.



Branched: Trust the Process

Pro

- Encourages clean/working master.

Con

- Encourages, but does not *require* code review.
- Must give explicit write permission to all team members.

This is the default strategy for private code repositories with named team members. For BIG projects, it can be time consuming to assign permissions to all devs.

Examples of Branched Workflow?

Internal projects with trusted developers

Internal projects using a centralized system (e.g. Git, Hg, bzt) **OR** centralized systems with liberal branching.

Review

- **Centralized** - Trust Everyone
- **Patched** - Trust No One; Propose *a Solution*
- **Forked** - Trust No One; Show *Your Work*
- **Branched** - Trust the Process

So What?

- 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

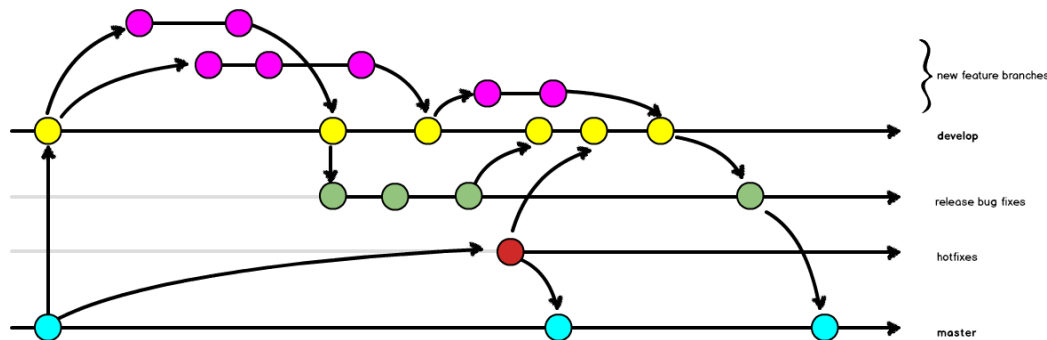
Identify and describe how your code is collated within your repository.

Branching Strategies

- Scheduled Release: [Gitflow](http://nvie.com/posts/a-successful-git-branching-model/)
[<http://nvie.com/posts/a-successful-git-branching-model/>]
or [Simplified Gitflow](http://drewfradette.ca/a-simpler-successful-git-branching-model/)
[<http://drewfradette.ca/a-simpler-successful-git-branching-model/>]
- Continuous Deployment: [Branch Per Feature](https://www.acquia.com/blog/pragmatic-guide-branch-feature-git-branching-strategy)
[<https://www.acquia.com/blog/pragmatic-guide-branch-feature-git-branching-strategy>]
or [GitHub Flow](http://scottchacon.com/2011/08/31/github-flow.html)
[<http://scottchacon.com/2011/08/31/github-flow.html>]

Scheduled Release

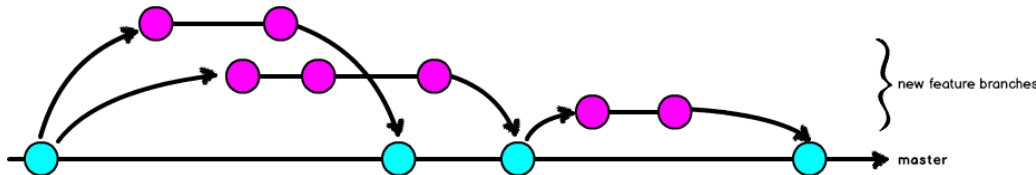
- Optimized for the collation of many smaller changes into a single release.
- Typically used for a download-able product; or web site with a scheduled release cycle (e.g. "Wednesdays").
- Incorporates human-reviews, and possibly automated tests.



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 code/developers and want to have a separate QA period. Thinking like a download-able product: version 4 vs. version 5 of The Software (a piece of software)

Continuous Deployment

- Code is deployed faster than scheduled releases; assumes all check-ins are deployable.
- Requires (trusted) test coverage.
- Typically uses a mechanical gatekeeper (CI) 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 multiple 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. CI, CD vs CD:

<http://puppetlabs.com/blog/continuous-delivery-vs-continuous-deployment-whats-diff>

Activity

Which best describes your current setup?

- Scheduled Release: [Gitflow](http://nvie.com/posts/a-successful-git-branching-model/)
[<http://nvie.com/posts/a-successful-git-branching-model/>]
or [Simplified Gitflow](http://drewfradette.ca/a-simpler-successful-git-branching-model/)
[<http://drewfradette.ca/a-simpler-successful-git-branching-model/>]
- Continuous Deployment: [Branch Per Feature](https://www.acquia.com/blog/pragmatic-guide-branch-feature-git-branching-strategy)
[<https://www.acquia.com/blog/pragmatic-guide-branch-feature-git-branching-strategy>]
or [GitHub Flow](http://scottchacon.com/2011/08/31/github-flow.html)
[<http://scottchacon.com/2011/08/31/github-flow.html>]

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.

So What?

- If you choose **SCHEDULED RELEASE**, streamline how your code is collated for release.
- If you choose **CONTINUOUS DEPLOYMENT**, codify how trust is deployed in your code.

Part 3

Commit Granularity

The Great Rebase Debate

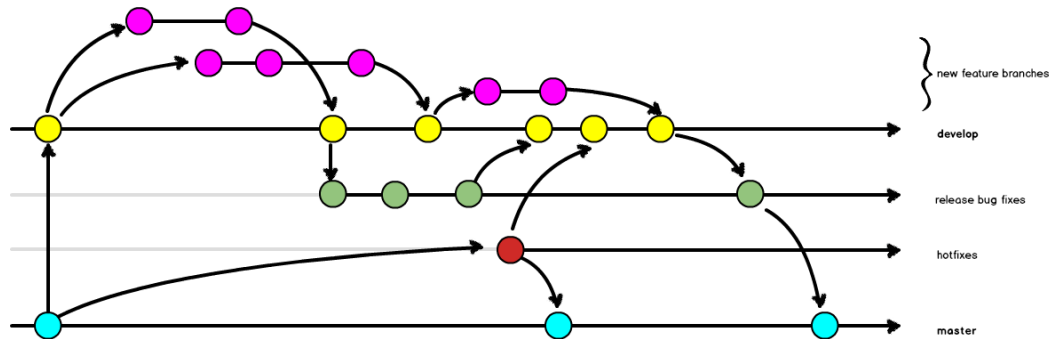
super nerdy rant alert!

Evolution of Social Coding

[<http://developerworkflow.com/resources/evolution-social-coding.html>]

What is a Commit

A record of the changes to the repository.



let's start with the very, very basics. A commit is a record of change.

How can we use Commits

- log
- gitk
- blame
- bisect

We use commits when we look at our project's history. We also use commits to debug our code with "advanced" tools, such as 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.

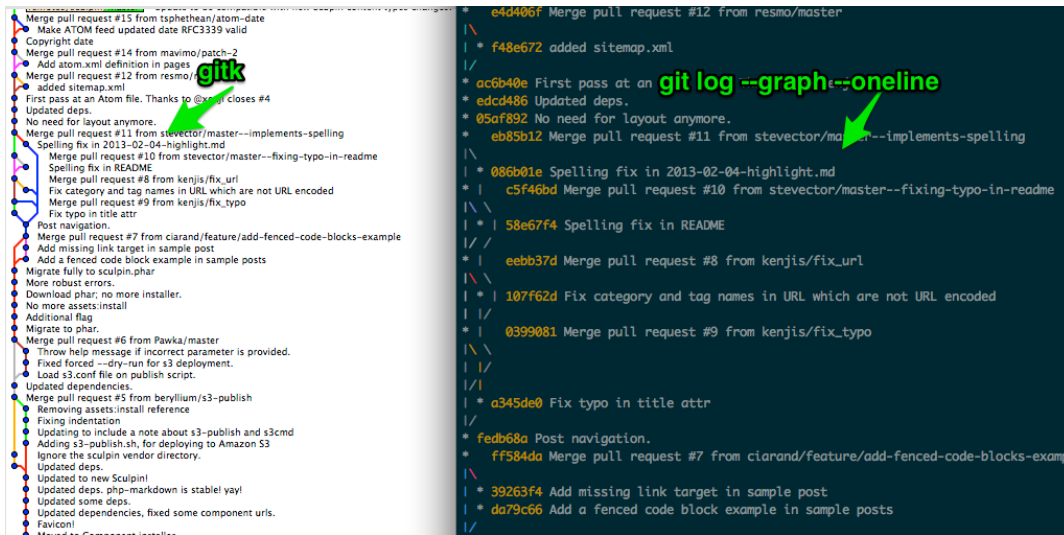
In other words: a commit is a whole idea.

the Linux kernel developers chose to use a patching workflow and created command line tools to support this branching strategy.

Sharing Work: Today

git push

Shares an entire branch, with all your micro commits.



```
gitk
Merge pull request #15 from tsphetean/atom-date
Make ATOM feed updated date RFC3339 valid
Copyright date
Merge pull request #14 from mavimo/patch-2
Add atom.xml definition in pages
Merge pull request #12 from resmo/master
added sitemap.xml
First pass at an Atom file. Thanks to @ymatthys closes #4
Updated deps.
No need for layout anymore.
Merge pull request #11 from stevector/master--implements-spelling
Spelling fix in 2013-02-04-highlight.md
Merge pull request #10 from stevector/master--fixing-typo-in-readme
Spelling fix in README
Merge pull request #8 from kenjis/fix_url
Fix category and tag names in URL which are not URL encoded
Merge pull request #9 from kenjis/fix_typo
Fix typo in title attr
Post navigation.
Merge pull request #7 from ciarand/feature/add-fenced-code-blocks-example
Add missing link target in sample post
Add a fenced code block example in sample posts
Migrate fully to sculpin.phar
More robust errors.
Download phar; no more installer.
No more assets:install
Additional flag
Migrate to phar.
Merge pull request #6 from Pawka/master
Throw help message if incorrect parameter is provided.
Fixed forced --dry-run for s3 deployment.
Load s3.conf file on publish script.
Updated dependencies.
Merge pull request #5 from beryllium/s3-publish
Removing assets:install reference
Fixing indentation
Updating to include a note about s3-publish and s3cmd
Adding s3-publish.sh, for deploying to Amazon S3
Ignore the sculpin vendor directory.
Updated deps.
Updated to new Sculpin!
Updated deps. php-markdown is stable yay!
Updated some deps.
Updated dependencies, fixed some component urls.
Favicon!
Moved to Composer installer

e4d406f Merge pull request #12 from resmo/master
f48e672 added sitemap.xml
ac6b40e First pass at an
edcd486 Updated deps.
05af892 No need for layout anymore.
eb85b12 Merge pull request #11 from stevector/master--implements-spelling
086b01e Spelling fix in 2013-02-04-highlight.md
c5f46bd Merge pull request #10 from stevector/master--fixing-typo-in-readme
58e67f4 Spelling fix in README
eebb37d Merge pull request #8 from kenjis/fix_url
107f62d Fix category and tag names in URL which are not URL encoded
0399081 Merge pull request #9 from kenjis/fix_typo
a345de0 Fix typo in title attr
fedb68a Post navigation.
ff584da Merge pull request #7 from ciarand/feature/add-fenced-code-blocks-example
39263f4 Add missing link target in sample post
da79c66 Add a fenced code block example in sample posts
```

Compare: bzz

The screenshot shows the bzz GUI window titled "Log - /Users/emmajane/Desktop/bzz/bzz.dev". The main panel displays a list of revisions with their commit messages, dates, and authors. A green arrow points to the revision number "6596" in the list, with a text overlay: "revision numbers for commits with branch depth; not commit hashes". Another green arrow points to the revision "6591.2.1", with a text overlay: "branches shown as summaries by default; click to open and show individual commits".

Rev	Commit Message	Date	Author
6596	(vila) Bzz config should save the changes explicitly when needed (Vincent)	2014-06-20 11:24	Patch Queue Manager
6597	(richard-wilbur) Jelmer: Don't pass blob to file.writelines(),	2014-05-07 22:56	Patch Queue Manager
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
6591	(vila) The XDG Base Directory Specification uses the XDG_CONFIG_HOME	2014-02-12 13:22	Patch Queue Manager
6590	(vila) Fix test failure on recent trusty kernels (the failure can't be	2014-02-10 05:02	Patch Queue Manager
6589	(vila) Stricter	2014-02-07 13:04	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

The detailed view for revision 6596 shows the following information:

- Revision: 6596 revid:pqm@pqm.ubuntu.com-20140507222027-mne60p2viqptfcmz
- Parents: 6595: (vila) Fix command line override handling for acceptable ke...
- Children: 6597: (richard-wilbur) Jelmer: Don't pass blob to file.writelines...
- Date: 2014-05-07 6:20 PM
- Committer: Patch Queue Manager <pqm@pqm.ubuntu.com>
- Branch: +trunk

The file list on the right shows the following files:

- bzrlib/config.py
- bzrlib/tests/test_config.py
- doc/developers/xdg_config_spec.txt
- doc/en/release-notes/bzz-2.7.txt

Buttons at the bottom include "Diff", "Refresh", and "Close".

branches are collapsed by default; there is a sane commit message when the branch is merged into master (unlike git which gives you a default "merged!" message)

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.

/rant

Evolution of Social Coding

[<http://developerworkflow.com/resources/evolution-social-coding.html>]

So What?

Discuss with your team how they want to find bugs, and therefore **HOW your commits should be recorded**.

are you social coding? Or are you using git as it was designed to work with the command line tools it ships with?

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

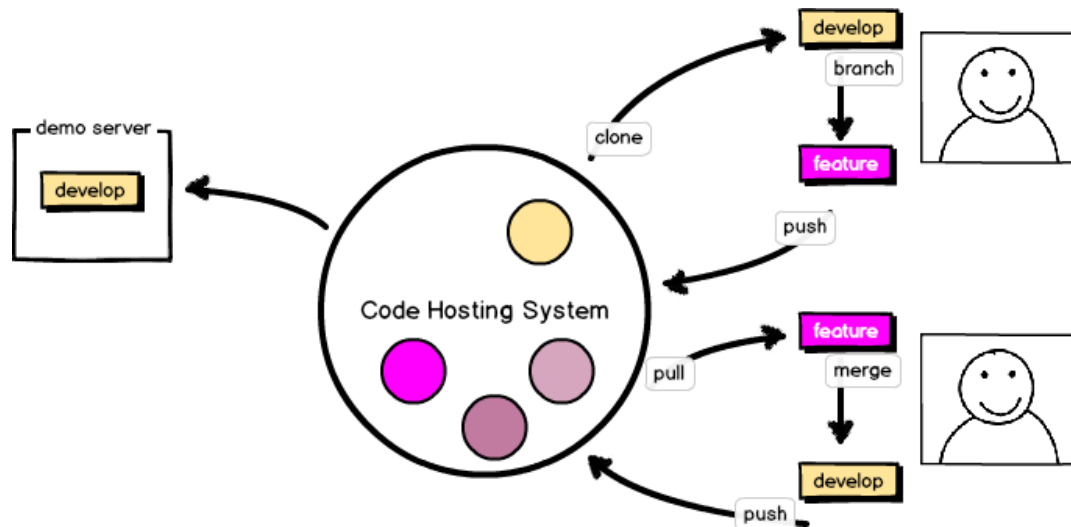
Project Highlights

- 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](#)
[\[../resources/workflow-sample-starwars.md\]](#)

Some Notes on Naming

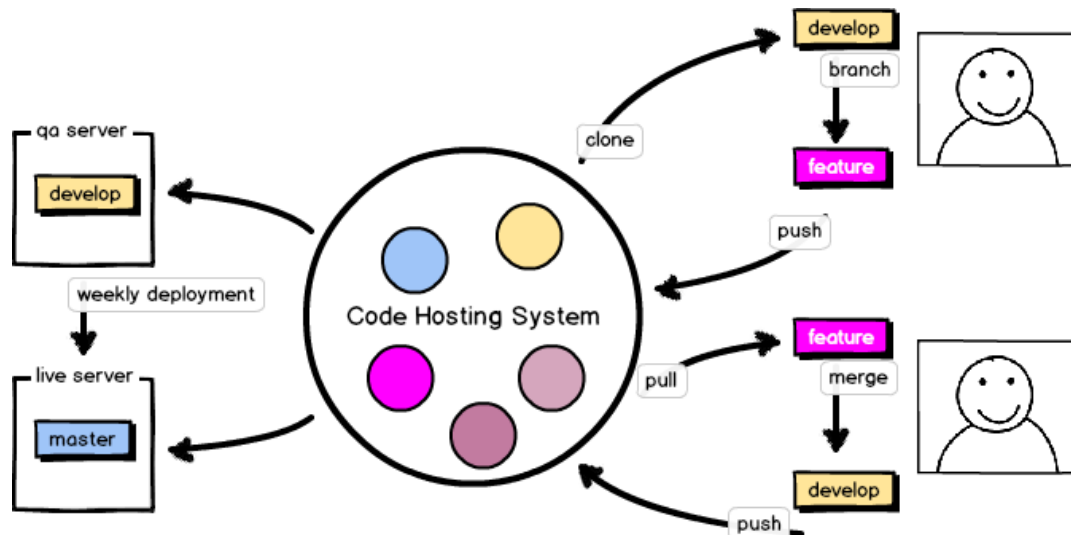
- Use terms which resonate with your team (MVP -> LBB).
- Giving a descriptive name to projects and processes allows you to change the meaning by changing the name.
- There are a lot of Ewoks.
- There are more My Little Ponies.

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.

Whispering Pines Workflow Documentation

github.com/emmajane/gitforteams

- [Whispering Pines Weekly Workflow](#)
[\[../resources/workflow-sample-whisperingpines-code.md\]](#)
- [Release philosophy](#)
[\[../resources/workflow-sample-whisperingpines-releasecycle.md\]](#)
- [Deployment](#)
[\[../resources/workflow-sample-whisperingpines-deployment.md\]](#)

Penultimate Activity: Sketch Your Workflow

- Restructure your previous diagrams to include the intrastate 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.)

Last Thing

Index Card

- What did you learn?
- What is (still) confusing?
- What will you change for your project?

Online

- Give the OSCON organizers (and me) your feedback.
- bit.ly/oscon-feedback-git

Resources

- [Developer Workflow](http://www.developerworkflow.com/)
[<http://www.developerworkflow.com/>]
- Scheduled Release: [Gitflow](http://nvie.com/posts/a-successful-git-branching-model/)
[<http://nvie.com/posts/a-successful-git-branching-model/>]
([Cheatsheet](http://danielkummer.github.io/git-flow-cheatsheet/)
[<http://danielkummer.github.io/git-flow-cheatsheet/>]
) or [Simplified Gitflow](http://drewfradette.ca/a-simpler-successful-git-branching-model/)
[<http://drewfradette.ca/a-simpler-successful-git-branching-model/>]
- Continuous Deployment: [Branch Per Feature](https://www.acquia.com/blog/pragmatic-guide-branch-feature-git-branching-strategy)
[<https://www.acquia.com/blog/pragmatic-guide-branch-feature-git-branching-strategy>]
or [GitHub Flow](http://scottchacon.com/2011/08/31/github-flow.html)
[<http://scottchacon.com/2011/08/31/github-flow.html>]
- Coming "soon": Git for Teams of One or More...the book!

Thanks!

Let's stay in touch!

[@emmajanehw](#)

[\[http://twitter.com/emmajanehw\]](http://twitter.com/emmajanehw)

emma@emmajane.net

<https://github.com/emmajane/gitforteam5>

[\[https://github.com/emmajane/gitforteam5\]](https://github.com/emmajane/gitforteam5)