Git Best Practice: Commit History

Karen Chan

December 2012

In this session we will talk about:

- Why it is good to have a clean and linear history
- What is a good commit message
- What is a clean history
- How to read the git log graph
- How to avoid merge commits
- How to clean up your commits / rewrite history

Why is it good to have a clean and linear history

- So we can read and understand what has been changed just by reading git log,
- Makes it easier to debug which commit caused a bug,
- Makes it easier to rollback or cherry-pick changes
- With a good commit message we can remember why something was changed

What is a good commit message

```
One sentence summary of what you did

More details about what you have done, for example why this
change exists, what you have deleted and why (which you can't
put as a comment in the code)
```

It is easier to rely on just git commit messages instead of tickets, so these commit messages are too brief:

- `12345 Add tests`
- `23456 Typo`
- `34567 Fixed bug`

In fact if you include everything, then the ticket number becomes much less important.

What is (not) a clean history

1. `Add comment validation for MADA submissions`
2. `Remove debug statements that were in the last commit`
3. `Fix typo`
4. `One of the tests failed, fixed it`
5. `Forgot to add the test files`

Do you really care about commits 2 - 5?
Is it useful for you to see all the mistakes I have made?

But I want to commit regularly...

Don’t worry, git gives you the ability to commit whenever you want, but also to clean it up before you publish your changes.

Git lets you rewrite history!

A linear history

Just having clean commits isn't really enough, having lots of merges also make the history look crazy:

```
git log --oneline --decorate --graph origin/production
```

Once upon a time, when history was linear

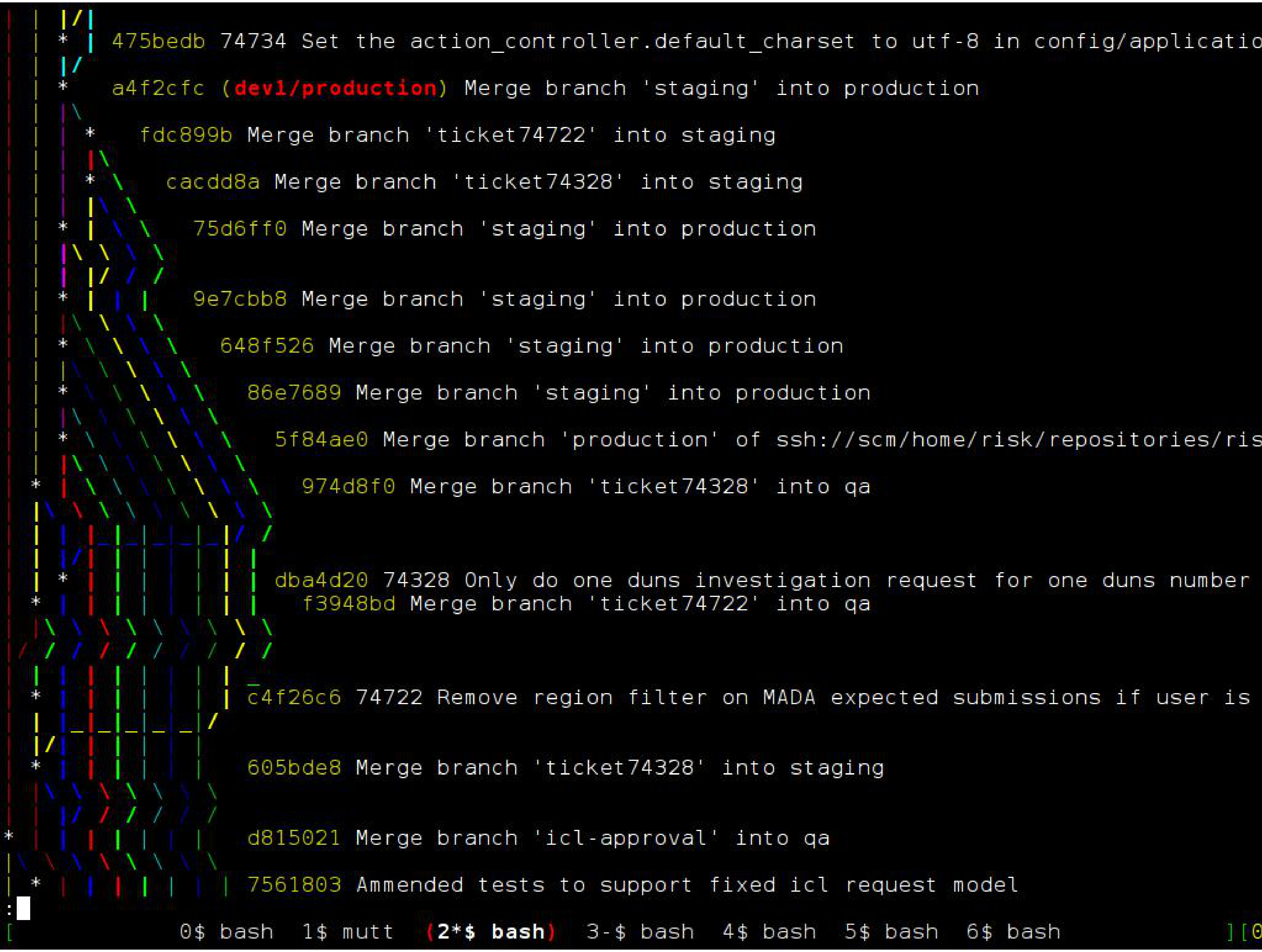
```
* 33784f7 Fixing the build (same packet...
* 624935a Converting null amounts to zero when...
* 7c42fd7 Changing deployment recipes to use git
* 3592704 [Ed] added empty data folders
```

A simple merge that we can still understand

```
* bbbc5ef Merge branch 'master' of svn.groupm.local...
|\
| * 60763cb Story #198:  CRUD pages ...
* | afe2440 changing column to bigint
* | 7ae049f Adding new table for duns...
* | 3c1047f Renaming InvestigationService...
|/
* 6e04551 Finishing refactoring to add product type to...
```

Getting complicated now...

```
* a4f2cfc (dev1/production) Merge branch ‘staging’ into...
|\
| * fdc899b Merge branch ‘ticket74722’ into staging
| |\
| | * c4f26c6 74722 Remove region filter on MADA expected...
| * | cacdd8a Merge branch ‘ticket74328’ into staging
| |\ \
| | |/
| |/|
| | * dba4d20 74328 Only do one duns investigation request...
* | | 75d6ff0 Merge branch ‘staging’ into production
```



```
| | |/|
| | * | 475bedb 74734 Set the action_controller.default_charset to utf-8 in config/applicatio
| | |/
| | *   a4f2cfc (dev1/production) Merge branch 'staging' into production
| | |\
| | | *   fdc899b Merge branch 'ticket74722' into staging
| | | |\
| | | * \   cacdd8a Merge branch 'ticket74328' into staging
| | | |\ \
| | * | \ \   75d6ff0 Merge branch 'staging' into production
| | |\ \ \ \
| | | |/ / /
| | * | | |   9e7cbb8 Merge branch 'staging' into production
| | |\ \ \ \
| | * \ \ \ \   648f526 Merge branch 'staging' into production
| | |\ \ \ \ \
| | * \ \ \ \ \   86e7689 Merge branch 'staging' into production
| | |\ \ \ \ \ \
| | * \ \ \ \ \ \   5f84ae0 Merge branch 'production' of ssh://scm/home/risk/repositories/ris
| | |\ \ \ \ \ \ \
| * | \ \ \ \ \ \ \   974d8f0 Merge branch 'ticket74328' into qa
| |\ \ \ \ \ \ \ \ \
| | | |_|_|_|_|_|/ /
| | |/| | | | | | |
| | * | | | | | | | dba4d20 74328 Only do one duns investigation request for one duns number
| * | | | | | | | |   f3948bd Merge branch 'ticket74722' into qa
| |\ \ \ \ \ \ \ \ \
|/ / / / / / / / / /
| | | | | | | | |  _
| * | | | | | | | c4f26c6 74722 Remove region filter on MADA expected submissions if user is
| | |_|_|_|_|_|/
| |/| | | | | |
| * | | | | | |   605bde8 Merge branch 'ticket74328' into staging
| |\ \ \ \ \ \ \
| | |/ / / / / /
* | | | | | | |   d815021 Merge branch 'icl-approval' into qa
|\ \ \ \ \ \ \ \
| * | | | | | | | 7561803 Ammended tests to support fixed icl request model
:
[            0$ bash  1$ mutt  (2*$ bash)  3-$ bash  4$ bash  5$ bash  6$ bash              ][0
```

Merge commits

The more layers we have the more merge commits we have. Usually merge commits are empty, and are not that useful to anyone.

A few of them is fine, but when there are merge commits every 2 actual commits it is useless noise and is distracting.

How to avoid merge commits when using git pull

- Do not use `--no-ff` when merging
  (`[merge] ff = true` in .gitconfig)

- Use `rebase` if you have changes on your branch
  (`[pull] rebase = true` in .gitconfig)

- Use `git push origin HEAD` to push only the current branch
  (`[push] default = current` in .gitconfig)

How to clean up your commits / rewrite history

- `git commit --amend`
- `git rebase`
- `git rebase -i`
- `git add -p`

git commit --amend

This lets you amend your last commit.
This is one of the most useful commands in git.

For example, you just committed, then you realized you forgot to add a file:

```
git add missing_file
git commit --amend
```

There, you just added missing_file to the last commit instead of doing a commit like "Forgot to add missing_file"

git rebase (1)

Rebasing is really important to keep a linear history.

For example, if I create a branch off `qa`, then someone pushes one commit to `qa`, then I commit to my branch, the commit history will look something like this:

```
| * 60763cb My commit
* | 3c1047f Commit 2
|/
* 6e04551 Commit 1
```

git rebase (2)

If you now merge into `qa` instead of doing `git rebase qa` first, the `qa` history will be like this:

```
* Merge branch ‘my-branch’
| \
| * 60763cb My commit
* | 3c1047f Commit 2
|/
* 6e04551 Commit 1
```

git rebase (3)

If you do git rebase qa, the qa history will be linear because you commit will go on top of the latest commit on qa:

```
* 60763cb My commit
|
* 3c1047f Commit 2
* 6e04551 Commit 1
```

You have avoided a not very useful “Merge my-branch into qa” commit.

git rebase (4)

What if I have a few branches that need to go back to qa, and I don't want to merge one then rebase the next one etc?

```
git cherry-pick qa..second-branch
git cherry-pick qa..third-branch
```

Cherry-pick will take your changes and commit message and apply to the current branch (`qa`), it won't remember what the parent of the commit is.

Which means qa is going to have a linear history.

git rebase -i (1)

This can be used to edit commit messages, reorder commits, execute tests in between commits etc.
Basically this is *THE* command to use for cleaning up history.

It is usually run like this: `git rebase -i origin/qa`

Your text editor will show something like this:

git rebase -i (2)

```
pick a2b1945 59860 Add turnover submitter and...
pick 795730e 59862 Create Turnover Declarations...
pick b52827a 59862 Add tests for...

# Rebase e7af698..731c2cc onto e7af698
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x, exec = run command (the rest of the line) using shell
```

The top commit will be the first to apply, all the way down to the last.

git add -p

This is very useful for splitting up your change into different commits.

```
git add -p README.txt
```

Allows you to edit the diff and lets you stage a subset of the changes you have made.

`git diff --cached`
`git commit` (no `-a`)

This lets you commit what you have chosen and leave other changes for another commit later.

A note on rewriting commit history

This should obviously only be done on your private local development branch.

Changing what is already on `origin/qa` for example, is not really ok.

In fact, git usually gives you a warning when you do `git push`:

```
To ssh://risk@scm/home/risk/repositories/riskplatform.git
 ! [rejected]        HEAD -> qa (non-fast-forward)
error: failed to push some refs to ‘ssh://risk@scm/home/risk/...’
```

Links

These articles have more about git rebase, merge and commit message:

- `http://sandofsky.com/blog/git-workflow.html`
- `http://randyfay.com/content/rebase-workflow-git`
- `http://darwinweb.net/articles/the-case-for-git-rebase`
- `http://www.reviewboard.org/docs/codebase/dev/git/clean-commits/`