Mastering git

Sebastian Feldmann

CHECK24

Mastering git

- Dos & Don’ts
- Tips & Tricks
- Hooks

Commit Messages

whatthecommit.com

- Major fixup.
- Fixed tpyo
- One does not simply merge into master
- Best commit ever

Rules

- Separate subject from body with one blank line
- Limit the subject line to 50 characters
- Do not end the subject line with a period
- Capitalize the subject line
- Wrap the body at 72 characters
- Use the body to explain what and why vs. how

Subject Line

Make 'getCommandLine' dependency explicit by adding the abstract meth... ··

sebastianfeldmann committed on 21 Feb ✓

6513235

Subject Line

Use the imperative mood for the subject line like git is doing it

```
Merge branch 'myfeature'
```

Always complete the following:
If applied, this commit will *your subject line here*

Oder in Deutsch:
Dieser Commit *deine Betreff Zeile*

Subject Examples

- Fixed bug with Y
- Changing behavior of X
- More fixes for broken stuff
- Sweet new API methods

- Refactor subsystem X for readability
- Update getting started documentation
- Remove deprecated methods
- Prepare version 1.0.0

Full Example

```
Summarize changes in around 50 characters or less

More detailed explanatory text, if necessary. Wrap it to about 72
characters or so. In some contexts, the first line is treated as the
subject of the commit and the rest of the text as the body. The
blank line separating the summary from the body is critical (unless
you omit the body entirely); various tools like `log`, `shortlog`
and `rebase` can get confused if you run the two together.

Explain the problem that this commit is solving. Focus on why you
are making this change as opposed to how (the code explains that).
Are there side effects or other unintuitive consequenses of this
change? Here's the place to explain them.

Further paragraphs come after blank lines.

 - Bullet points are okay, too

 - Typically a hyphen or asterisk is used for the bullet, preceded
   by a single space, with blank lines in between, but conventions
   vary here

If you use an issue tracker, put references to them at the bottom,
like this:

Resolves: #123
See also: #456, #789
```

Tips & Tricks

Autocompletion

Download the completion script

```
curl -O https://raw.github.com/git/git/master/contrib/completion/git-completion.bash
mv git-completion.bash .git-completion.bash
```

Then in your `.bash_profile` add the following

```
if [ -f ~/.git-completion.bash ]; then
    . ~/.git-completion.bash
fi
```

.git Template

- Default /usr/local/share/git-core/templates
- Custom hooks
- Custom excludes

```
git config --global init.templatedir '~/.git_template/template'
```

OS Aliases

Use system aliases

```
# I’m lazy as hell
alias g='git'

# git status in a flash
alias gs='git status’
alias gss='git status -s’

# go to repository root directory
alias gr='[! -z `git rev-parse --show-cdup`] && cd `git rev-parse --show-cdup || pwd`'
```

git aliases

Add via terminal or edit ~/.gitconfig

```
$ git config --global alias.unstage "reset HEAD"
$ git config --global alias.wtf "log -p"
$ git config --global alias.l "log --oneline --graph"
$ git config --global alias.b "branch"
$ git config --global alias.c "commit"
$ git config --global alias.p "pull –rebase"
```

Reset Files

Return to a previous version and discard all changed

```
$ git reset --hard {{some-commit}}
```

Return to a previous version, changes are unstaged

```
$ git reset {{some-commit}}
```

Return to a previous version, changes are staged

```
$ git reset --soft {{some-commit}}
```

Log

`--autor="Sebastian"`	Only show commits made by a certain author
`--name-only`	Only show names of files that changed
`--oneline`	Show commit data compressed to one line
`--graph`	Show dependency tree for all commits
`--reverse`	Show commits in reverse order (Oldest commit first)
`--after`	Show all commits that happened after certain date
`--before`	Show all commits that happened before certain data
`--stat`	Show all commits with some statistics

Example

```
$ git log --author="Sebastian" --after="1 week ago" --oneline
```

git log

You can use the regular less command to search

```
/{{your-search-here}}
```

Use lower case **`n`** to navigate to the next occurrence and upper case **`N`** to the previous one.

Ignore Whitespace

You can easily ignore whitespace for diff and blame

```
$ git diff -w
$ git blame -w
```

git stash

Store uncommitted changes

```
$ git stash
```

Show stashed changes

```
$ git stash list
```

git stash

Apply latest stashed state and remove from list

```
$ git stash pop
```

Apply any stashed state and keep in list

```
$ git stash apply stash@{n}
```

Clean up the stash

```
$ git stash drop stash@{n}
$ git stash clear
```

Demo

Partial Add

add only some changes in a file

```
$ git add -p
```

Partial Add

y stage this hunk for the next commit

n do not stage this hunk for the next commit

d do not stage this hunk or any of the later hunks in the file

s split the current hunk into smaller hunks

e manually edit the current hunk

? print hunk help

Demo

git rebase

> *Ahh, but the bliss of rebasing isn't without its drawbacks, which can be summed up in a single line:*
>
> *Do not rebase commits that exist outside your repository*
>
> *If you follow that guideline, you'll be fine. If you don't, people will hate you, and you'll be scorned by friends and family.*

–git book

git rebase

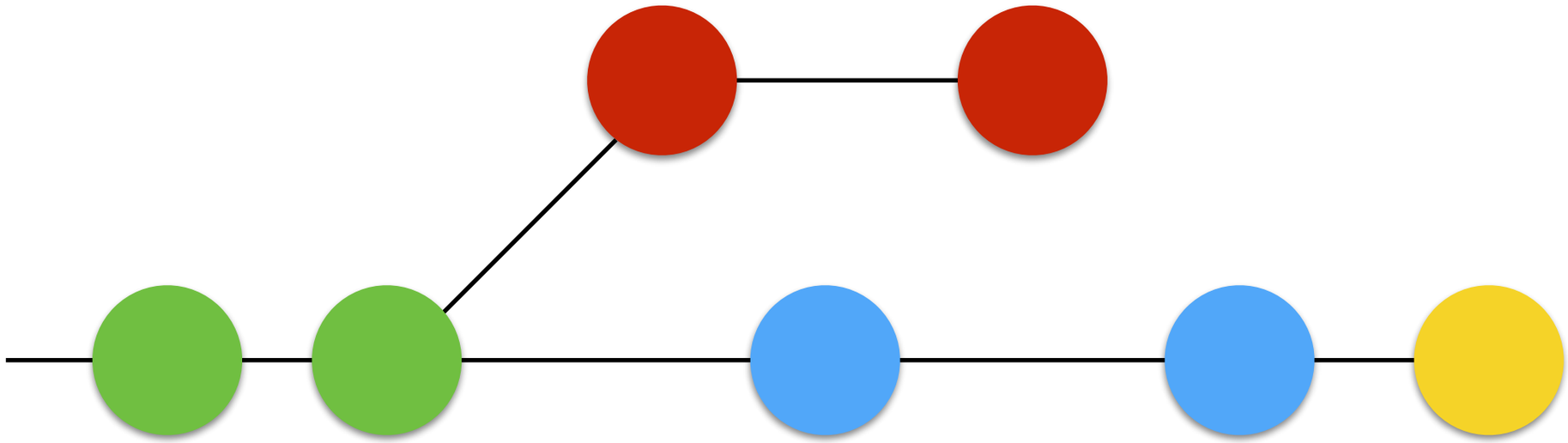
Committing on shared branches

> `Merge remote-tracking branch ‘origin/master’`

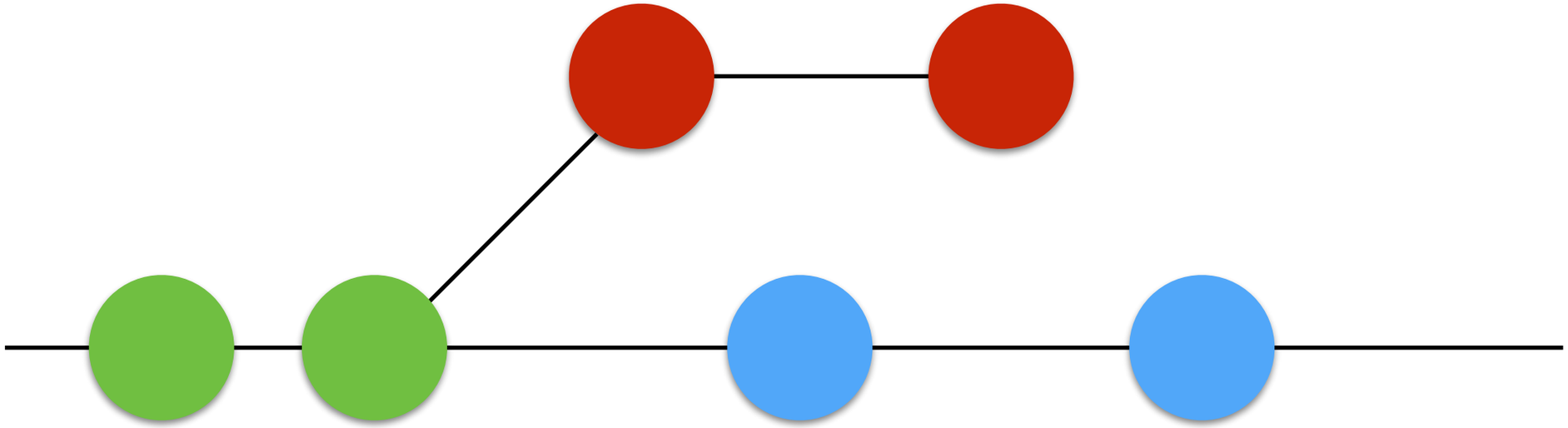
No big deal and completely safe, but still messes up the log history a bit.

```
$ git pull --rebase
```

git merge



git rebase

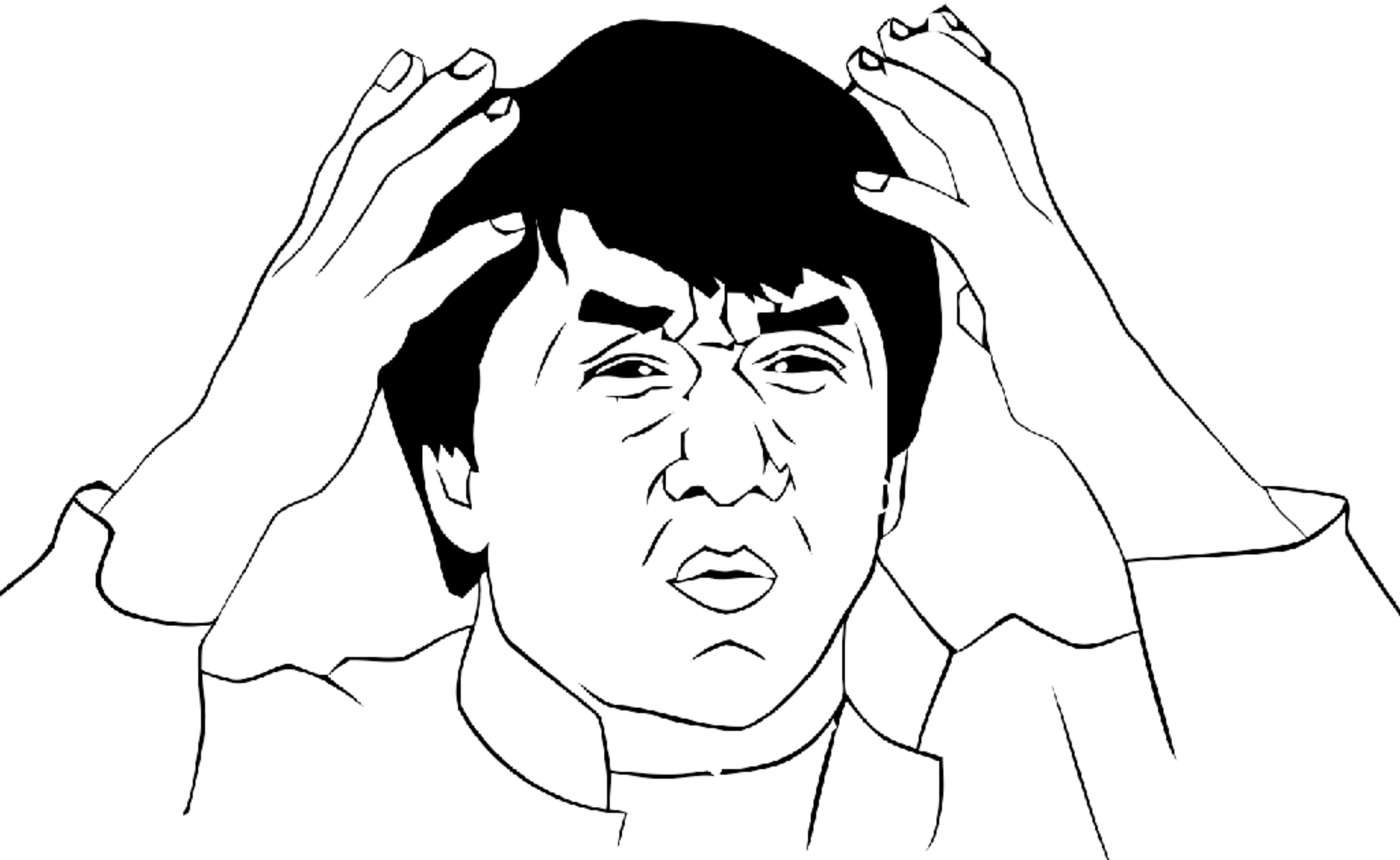


Demo

git amend

```
$ git add path/toFile/withCodingStandardFix.php
$ git commit --amend
```

someone broke my stuff



but when

git bisect

Find commits that break your code fast

```
$ git bisect start
$ git bisect good {{some-commit}}
$ git bisect bad HEAD
```

Test and flag the commit

```
$ git bisect good|bad
```

git bisect

Go back to the starting point

```
$ git bisect reset
```

Get a summary report of last bisect run

```
$ git bisect log
```

Demo

git hooks

Hooks

- post-checkout
- commit-msg
- pre-commit
- pre-push

CaptainHook

- Easy to use
- Shareable configuration
- Simple to extend

https://github.com/sebastianfeldmann/captainhook

Usage

- Install via composer

  `sebastianfeldmann/captainhook`

- Manage hooks via CLI

  `vendor/bin/captainhook`

Configuration

- JSON configuration
- captainhook.json

Extend

- Commit message "Rules"
- Custom Actions
- Static methods
- Commands

Demo

thank you

Please leave some feedback

Sebastian Feldmann

phpbu

https://phpbu.de

php User Group Munich

CHECK24

@movetodevnull

sebastianfeldmann

Q & A