
a p p
m o t
i o n

# Beyond “git commit”

Tipps und Tricks für Git-Ninjas

appmotion

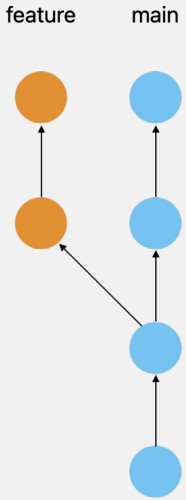
# Agenda

- Merge Requests aktualisieren
- Commits auf Branches anwenden
- „Shit, I lost my work“
- History durchsuchen
- Binäre Suche nach Bugs
- Honorable Mentions

# Wie aktualisiere ich einen Merge Request?

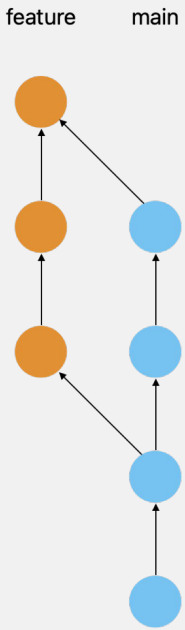
git-rebase

# Klassischer Merge

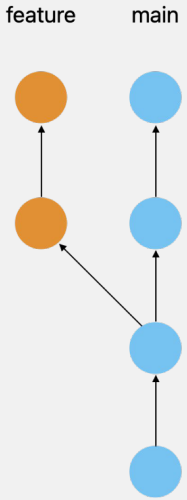


# Klassischer Merge

git merge main

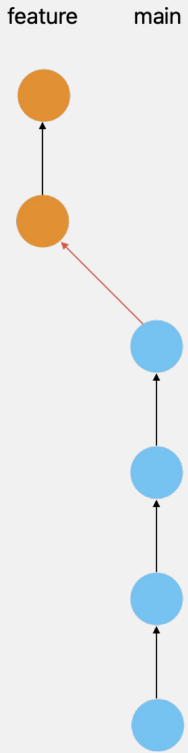


# Rebase



# Rebase

git rebase main



# Wie räume ich einen Merge Request auf?

git-rebase -i

# git rebase -i main

```
pick 164aaef Add feature Y
pick dc0bc36 Fix typo
pick ce5bc54 Fix feature Y
pick 20a82ef Fix CI
pick 31ad9f3 Actually fix CI

# Rebase von 9006d6a..31ad9f3 auf 9006d6a (5 Kommandos)
#
# Befehle:
# p, pick <Commit> = Commit verwenden
# r, reword <Commit> = Commit verwenden, aber Commit-Beschreibung bearbeiten
# e, edit <Commit> = Commit verwenden, aber zum Nachbessern anhalten
# s, squash <Commit> = Commit verwenden, aber mit vorherigem Commit vereinen
# f, fixup [-C | -c] <Commit> = wie "squash", aber nur die vorherige
#                    Commit-Beschreibung behalten, außer -C wird verwendet,
#                    in diesem Fall wird nur diese Commit-Beschreibung behalten;
#                    -c ist das Gleiche wie -C, aber ein Editor wird geöffnet
# x, exec <Commit> = Befehl (Rest der Zeile) mittels Shell ausführen
# b, break = hier anhalten (Rebase später mit 'git rebase --continue' fortsetzen)
# d, drop <Commit> = Commit entfernen
# l, label <Label> = aktuellen HEAD mit Label versehen
# t, reset <Label> = HEAD zu einem Label umsetzen
# m, merge [-C <Commit> | -c <Commit>] <Label> [# <eineZeile>]
#         Merge-Commit mit der originalen Merge-Commit-Beschreibung erstellen
#         (oder die eine Zeile, wenn keine originale Merge-Commit-Beschreibung
#         spezifiziert ist). Benutzen Sie -c <Commit> zum Bearbeiten der
#         Commit-Beschreibung.
# u, update-ref <Referenz>= einen Platzhalter für die zu aktualisierende <Referenz>
#                       zu dieser Position in den neuen Commits erstellen. Die <Referenz>wird
#                       am Ende des Rebase aktualisiert
#
# Diese Zeilen können umsortiert werden; sie werden von oben nach unten
# ausgeführt.
#
# Wenn Sie hier eine Zeile entfernen, wird DIESER COMMIT VERLOREN GEHEN.
#
# Wenn Sie jedoch alles löschen, wird der Rebase abgebrochen.
```

# Wo ist ...?

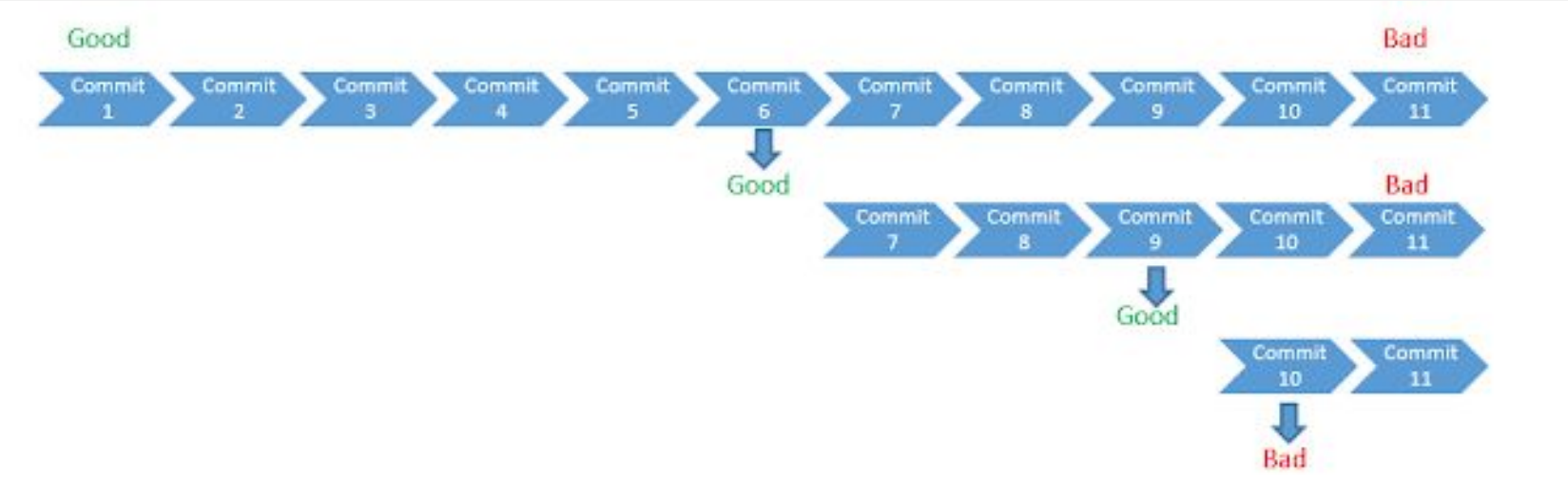
git log (-S)

# “Shit, I lost my work!”

git reflog

# Wo kommt der Bug her?

git bisect


Good
Bad
Commit 1
Commit 2
Commit 3
Commit 4
Commit 5
Commit 6
Commit 7
Commit 8
Commit 9
Commit 10
Commit 11
Good
Bad
Commit 7
Commit 8
Commit 9
Commit 10
Commit 11
Good
Commit 10
Commit 11
Bad

# Honorable Mentions

Aus: https://git-scm.com/docs/git#_git_commands

## NAME

git-whatchanged - Show logs with difference each commit introduces

## SYNOPSIS

```
git whatchanged <option>…
```

## DESCRIPTION

Shows commit logs and diff output each commit introduces.

New users are encouraged to use git-log[1] instead. The `whatchanged` command is essentially the same as git-log[1] but defaults to show the raw format diff output and to skip merges.

The command is kept primarily for historical reasons; fingers of many people who learned Git long before `git log` was invented by reading Linux kernel mailing list are trained to type it.

## Examples

`git whatchanged -p v2.6.12.. include/scsi drivers/scsi`
: Show as patches the commits since version **`v2.6.12`** that changed any file in the include/scsi or drivers/scsi subdirectories

`git whatchanged --since="2 weeks ago" -- gitk`
: Show the changes during the last two weeks to the file **`gitk`**. The "--" is necessary to avoid confusion with the **branch** named **`gitk`**

## NAME

git-notes - Add or inspect object notes

## SYNOPSIS

```
git notes [list [<object>]]
git notes add [-f] [--allow-empty] [-F <file> | -m <msg> | (-c | -C)
<object>] [<object>]
git notes copy [-f] ( --stdin | <from-object> [<to-object>] )
git notes append [--allow-empty] [-F <file> | -m <msg> | (-c | -C)
<object>] [<object>]
git notes edit [--allow-empty] [<object>]
git notes show [<object>]
git notes merge [-v | -q] [-s <strategy> ] <notes-ref>
git notes merge --commit [-v | -q]
git notes merge --abort [-v | -q]
git notes remove [--ignore-missing] [--stdin] [<object>…]
git notes prune [-n] [-v]
git notes get-ref
```

## DESCRIPTION

Adds, removes, or reads notes attached to objects, without touching the objects themselves.

By default, notes are saved to and read from `refs/notes/commits`, but this default can be overridden. See the OPTIONS, CONFIGURATION, and ENVIRONMENT sections below. If this ref does not exist, it will be quietly created when it is first needed to store a note.

A typical use of notes is to supplement a commit message without changing the commit itself. Notes can be shown by **`git log`** along with the original commit message. To distinguish these notes from the message stored in the commit object, the notes are indented like the message, after an unindented line saying "Notes (<refname>):" (or "Notes:" for `refs/notes/commits`).

## NAME

git-send-email - Send a collection of patches as emails

## SYNOPSIS

```
git send-email [<options>] <file|directory>…
git send-email [<options>] <format-patch options>
git send-email --dump-aliases
```

## DESCRIPTION

Takes the patches given on the command line and emails them out. Patches can be specified as files, directories (which will send all files in the directory), or directly as a revision list. In the last case, any format accepted by git-format-patch[1] can be passed to git send-email, as well as options understood by git-format-patch[1].

The header of the email is configurable via command-line options. If not specified on the command line, the user will be prompted with a ReadLine enabled interface to provide the necessary information.

There are two formats accepted for patch files:

1. mbox format files

   This is what git-format-patch[1] generates. Most headers and MIME formatting are ignored.

2. The original format used by Greg Kroah-Hartman's **`send_lots_of_email.pl`** script

   This format expects the first line of the file to contain the "Cc:" value and the "Subject:" of the message as the second line.

## NAME

git-instaweb - Instantly browse your working repository in gitweb

## SYNOPSIS

```
git instaweb [--local] [--httpd=<httpd>] [--port=<port>]
              [--browser=<browser>]
git instaweb [--start] [--stop] [--restart]
```

## DESCRIPTION

A simple script to set up `gitweb` and a web server for browsing the local repository.

## NAME

git-worktree - Manage multiple working trees

## SYNOPSIS

```
git worktree add [-f] [--detach] [--checkout] [--lock [--reason <string>]]
                 [-b <new-branch>] <path> [<commit-ish>]
git worktree list [-v | --porcelain [-z]]
git worktree lock [--reason <string>] <worktree>
git worktree move <worktree> <new-path>
git worktree prune [-n] [-v] [--expire <expire>]
git worktree remove [-f] <worktree>
git worktree repair [<path>…]
git worktree unlock <worktree>
```

## DESCRIPTION

Manage multiple working trees attached to the same repository.

A git repository can support multiple working trees, allowing you to check out more than one branch at a time. With `git worktree add` a new working tree is associated with the repository, along with additional metadata that differentiates that working tree from others in the same repository. The working tree, along with this metadata, is called a "worktree".

This new worktree is called a "linked worktree" as opposed to the "main worktree" prepared by git-init[1] or git-clone[1]. A repository has one main worktree (if it's not a bare repository) and zero or more linked worktrees. When you are done with a linked worktree, remove it with `git worktree remove`.


It's leviOsa.