Git for (not-so) dummies

Michaël Hauspie <michael.hauspie@univ-lille1.fr>

Université Lille 1 - IUT 'A' - CRIStAL

September 16, 2015

Outline

What is git ?

Short story

- From 1991 to 2002, linux kernel was developed by sharing and archiving patches,
- in 2002, project started to use BitKeeper,
- in 2005, free of charge use of BitKeeper for the development of the kernel was revoked,
- developers (and especially Linus Torvalds) started their own distributed version control system (DVCS).

VCS vs DVCS

Version control system

- A system to track and manage modification of source code,
- based on a central repository of the code,
- each developer has its own working tree and submit modification on the remote server.

Version control system

- A system to track and manage modification of source code,
- based on a central repository of the code,
- each developer has its own working tree and submit modification on the remote server.

Distributed version control system

- No needed central repository (although frequently used in common workflows),
- every developer has its own copy of the repository,
- modification are committed to the local repository,
- can use several repository to fetch patches from.

Version control system

- A system to track and manage modification of source code,
- based on a central repository of the code,
- each developer has its own working tree and submit modification on the remote server.

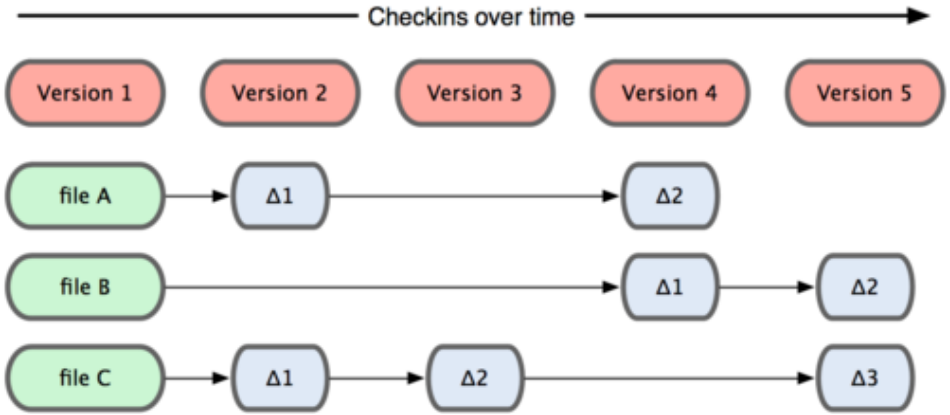
Distributed version control system

- No needed central repository (although frequently used in common workflows),
- every developer has its own copy of the repository,
- modification are committed to the local repository,
- can use several repository to fetch patches from.

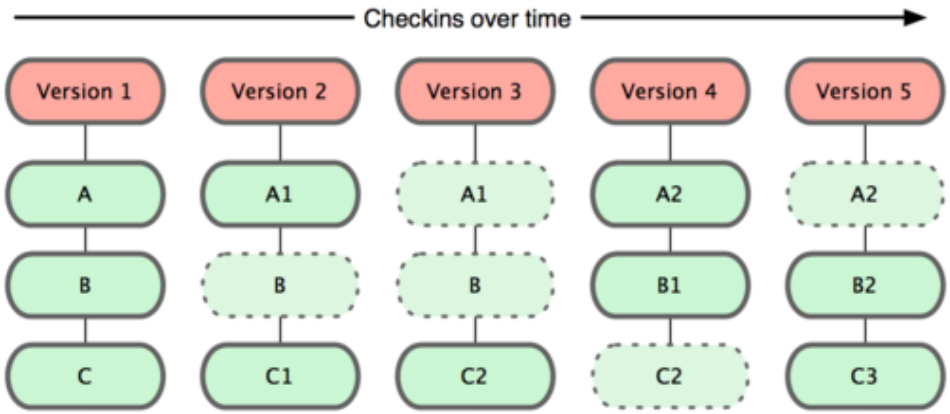
$\Rightarrow$ git is a distributed version control system

Git basics overview

Usually, VCS work with patches



Git acts as a mini filesystem



What is git ?

As it acts as a file systems, it comes with commands:

- to manipulate this file system directly → git plumbing commands,
- to use this file system to implement a user friendly DVCS → git porcelain commands

Every operation is local

- It is fast.
- It is very fast!
- You can work without being connected and still keep a precise commit history.

Every operation is local

- It is fast.
- It is very fast!
- You can work without being connected and still keep a precise commit history.

Every object stored is check-summed (content, metadata...)

- And referred by this checksum (SHA1),
- you can not alter an object without git knowing it

Every operation is local

- It is fast.
- It is very fast!
- You can work without being connected and still keep a precise commit history.

Every object stored is check-summed (content, metadata...)

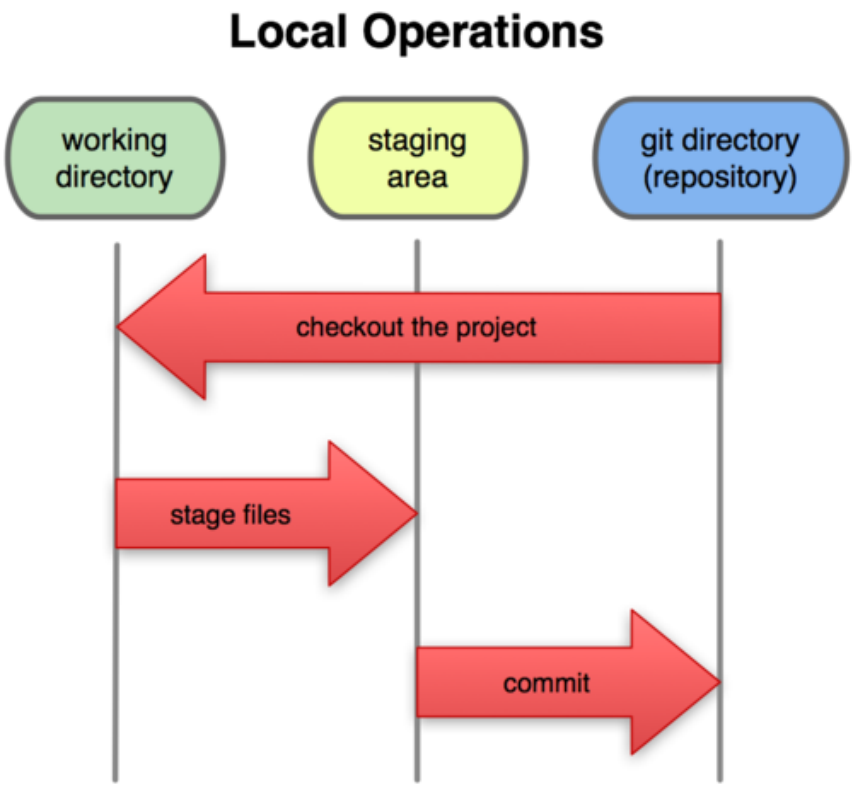
- And referred by this checksum (SHA1),
- you can not alter an object without git knowing it

Removing data is hard

- It is nearly impossible to loose committed data (except on purpose or hardware crash...),
- nearly all git operation adds something to the filesystem (even removing a file from the source tree).

The three states

- Your files can be in three different spaces:
  1. working directory,
  2. index (or staging area),
  3. repository.



Git directory

- Stores the repository meta-datas,
- the most important part, holds every single objects,
- this is wat you get when cloning a repository.

Git directory

- Stores the repository meta-datas,
- the most important part, holds every single objects,
- this is wat you get when cloning a repository.

Staging area

- Contains a snapshot of what will go in the next commit

Git directory

- Stores the repository meta-datas,
- the most important part, holds every single objects,
- this is wat you get when cloning a repository.

Staging area

- Contains a snapshot of what will go in the next commit

Working directory

- These are the files you are working on

Git commit workflow

1. Work on your files,
2. add them to the staging area,
3. commit.

Git commit workflow

1. Work on your files,
2. add them to the staging area,
3. commit.

Staging area?

- Can be used to split a bunch of work in multiple commits,
- can be by-passed easily if not needed.

Git commands

- Default use pattern of git command is `git <command> <command args>`
- Most commonly used commands:
  - `add`: Add file contents to the index
  - `branch`: List, create, or delete branches
  - `checkout`: Checkout a branch or paths to the working tree
  - `clone`: Clone a repository into a new directory
  - `commit`: Record changes to the repository
  - `diff`: Show changes between commits, commit and working tree, etc
  - `fetch`: Download objects and refs from another repository
  - `help`: Get help for a command (equivalent to `man git command`)
  - `init`: Create an empty git repository or reinitialize an existing one
  - `log`: Show commit logs
  - `merge`: Join two or more development histories together
  - `pull`: Fetch from and merge with another repository or a local branch
  - `push`: Update remote refs along with associated objects
  - `status`: Show the working tree status

Local repository

Creating a repository

git init

```
$ git init awesome-project
Initialized empty Git repository in awesome-project/.git/
```

git init

```
$ git init awesome-project
Initialized empty Git repository in awesome-project/.git/
```

Repository overview

```
$ tree awesome-project/.git
awesome-project/.git/
|-- HEAD
|-- branches
|-- config
|-- description
|-- hooks
|   |-- applypatch-msg.sample
|   |-- commit-msg.sample
|   |-- post-update.sample
|   |-- pre-applypatch.sample
|   |-- pre-commit.sample
|   |-- pre-rebase.sample
|   |-- prepare-commit-msg.sample
|   `-- update.sample
|-- info
|   `-- exclude
|-- objects
|   |-- info
|   `-- pack
`-- refs
    |-- heads
    `-- tags
```

Commit changes

First commit

We have to:

1. create a new file and fill it with some content,
2. add it to the stage area (*i.e stage it*) `git add [-p]`,
3. commit. `git commit`

First commit

We have to:

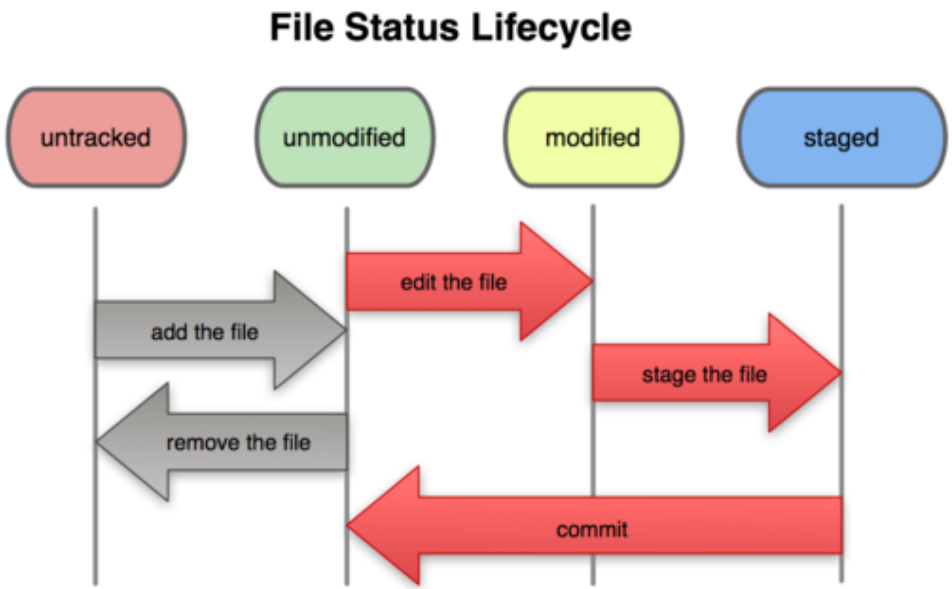
1. create a new file and fill it with some content,
2. add it to the stage area (*i.e stage it*) `git add [-p]`,
3. commit. `git commit`

Example:

```
$ echo 'This program implements the awesome-project [...]' > README
$ git add README
$ git commit
[master (root-commit) 535b20a] Initial commit, added README
 1 file changed, 1 insertion(+)
 create mode 100644 README
```

File status

1. untracked → not managed by git,
2. unmodified → up-to-date with git repository,
3. modified → modified since last commit, but will not be included in next commit,
4. staged → in the staging area, will be included in the next commit.



Modifying and committing

1. modify,
2. stage,
3. commit.

```
$ echo 'Implemented with awesomeness in mind!' >> README
$ git add README
$ git commit
[master dbe20b6] Awesomeness added!
 1 file changed, 1 insertion(+)
```

Listing commits

`git log`

```
$ git log
commit dbe20b6ce7135a7d2ad02e6d17e15a61c0efe5a0
Author: Michael Hauspie <michael.hauspie@lifl.fr>
Date:   Fri Apr 6 14:09:06 2012 +0200

    Awesomeness added!

commit 535b20ac57d1fb5fe294924d099822411fba7040
Author: Michael Hauspie <michael.hauspie@lifl.fr>
Date:   Fri Apr 6 14:08:40 2012 +0200

    Initial commit, added README
```

How to get information on file status

`git status`

```
$ git status
# On branch master
nothing to commit (working directory clean)

$ echo 'Will certainly be as fast as it will be awesome!' >> README

$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working director
#
# modified:   README
#
no changes added to commit (use "git add" and/or "git commit -a")
```

How to get information on file status

```
$ echo 'Michael Hauspie' > AUTHOR

$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working director
#
# modified:   README
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
# AUTHOR
no changes added to commit (use "git add" and/or "git commit -a")
```

How to get information on file status

```
$ git add AUTHOR

$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# new file:   AUTHOR
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working director
#
# modified:   README
#
```

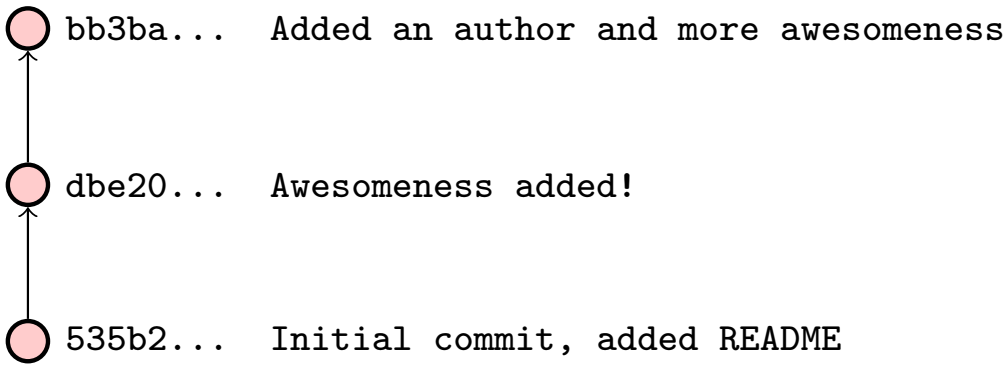
How to get information on file status

```
$ git add README

$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# new file:   AUTHOR
# modified:   README
#

$ git commit
[master bb3ba6c] Added an author and more awesomeness
 2 files changed, 2 insertions(+)
 create mode 100644 AUTHOR
```

References

Tiny bits of git internals

- all objects are identified (and can be referenced) by a SHA-1 checksum
- a commit object references a snapshot of the code (a tree object) in the git filesystem
- a commit object also references its parent(s) commit(s)



References

- Every git object is referenced by a SHA-1 checksum,
- remembering SHA-1 is hard,
- references are symbolic names for SHA-1,
- and are implemented as a simple text file storing the SHA-1.

```
$ find .git/refs -type f | grep refs
.git/refs/heads/master
$ cat .git/refs/heads/master
bb3ba6c929e2d9923779cb434c4eabc219cbc65b
```

Using reference names

- When a symbolic name is used, git tries to find a matching file depending on the context
  - in `.git/refs/heads` for a branch
  - in `.git/refs/tags` for a tag
  - in `.git/refs/remotes` for a remote
- You can always use an *absolute reference*
  - `refs/heads/master` instead of `master`,
  - `heads`, `tags`, and `remotes` have a particular meaning **but** you can use other kind of refs such as `refs/merge-requests/1` or `refs/my-awesome-patches/patch1`,
  - custom reference convention is often used by workflows, code review tools...

Branching and merging

Branching

- branching is a way to diverge the development from the main line,
- often referred to as **THE** killing feature of git,
- git branches are extremely lightweight,
  - are created instantly,
  - can be merged easily.

Branching

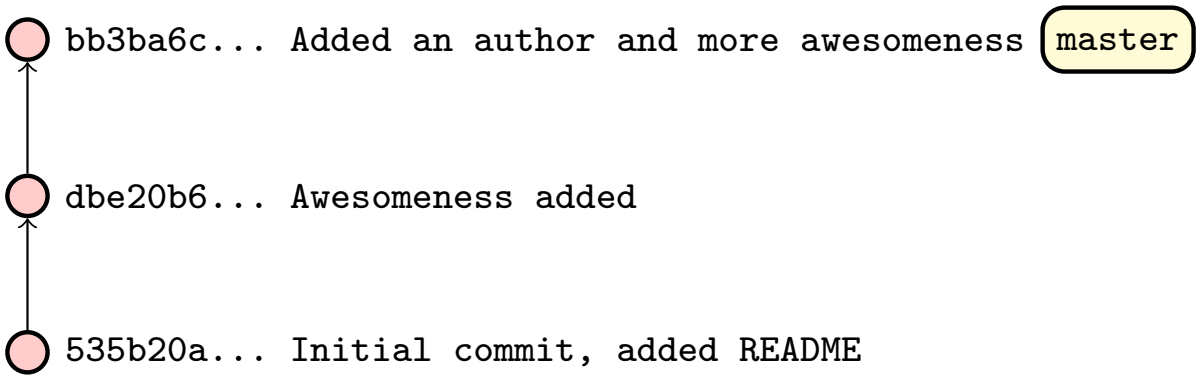
- branching is a way to diverge the development from the main line,
- often referred to as **THE** killing feature of git,
- git branches are extremely lightweight,
  - are created instantly,
  - can be merged easily.

How are they handled by git?

- A branch is only a pointer to a commit object,
- creating a branch is just about creating a single text file that contains a SHA-1 in the `.git/refs/heads` folder.

The master branch

- By default, the only branch of a repository is the master branch,
- a `git commit` creates a new commit object and updates the current branch so that it points to this commit object.



Using branches

The two main commands when using branches are

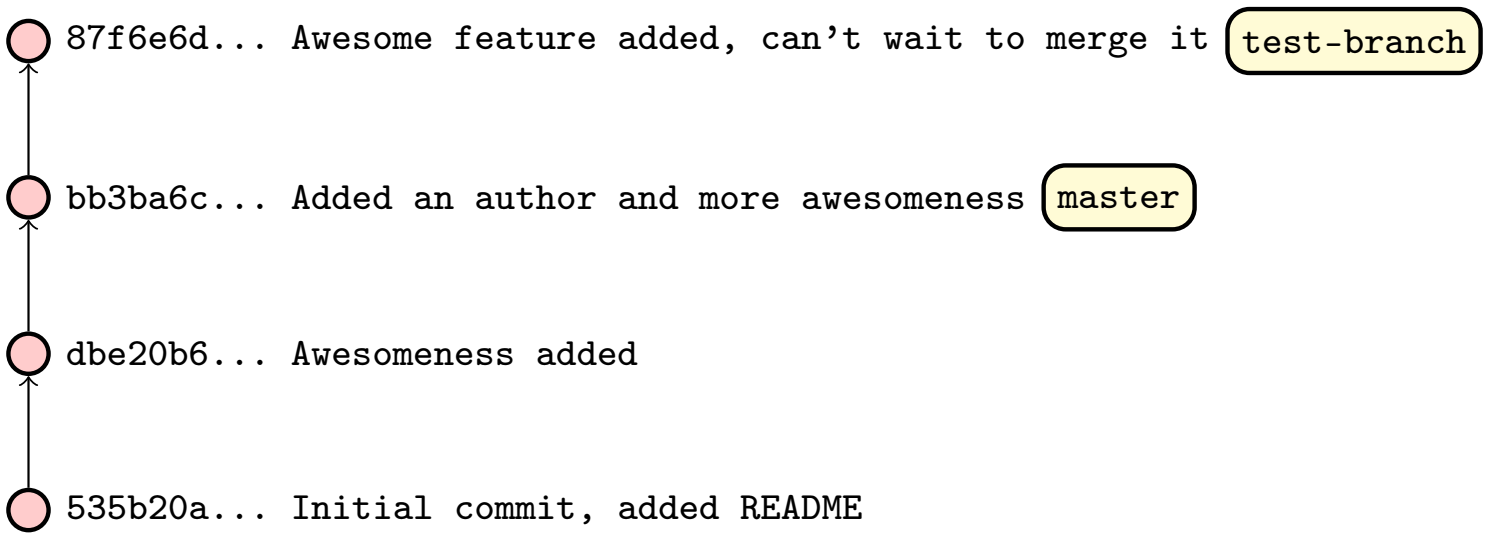
- `git branch` : manage branches (create, delete, modify upstream),
- `git checkout` : navigate through branches → modify your working directory to match a given branch

Branch creation

- `git branch <branch-name>` creates a new branch but does not switch to it (*i.e.* your working directory stays on the current branch),
- you must then use `git checkout <branch-name>` to work in the new branch,
- you can use `git checkout -b <branch-name>` to create the branch and switch to it.

Example

```
$ git checkout -b test-branch
$ echo 'More awesome features added >> README
$ git commit -a
[test-branch 87f6e6d] Awesome feature added, can't wait to merge it!
 1 file changed, 1 insertion(+)
```



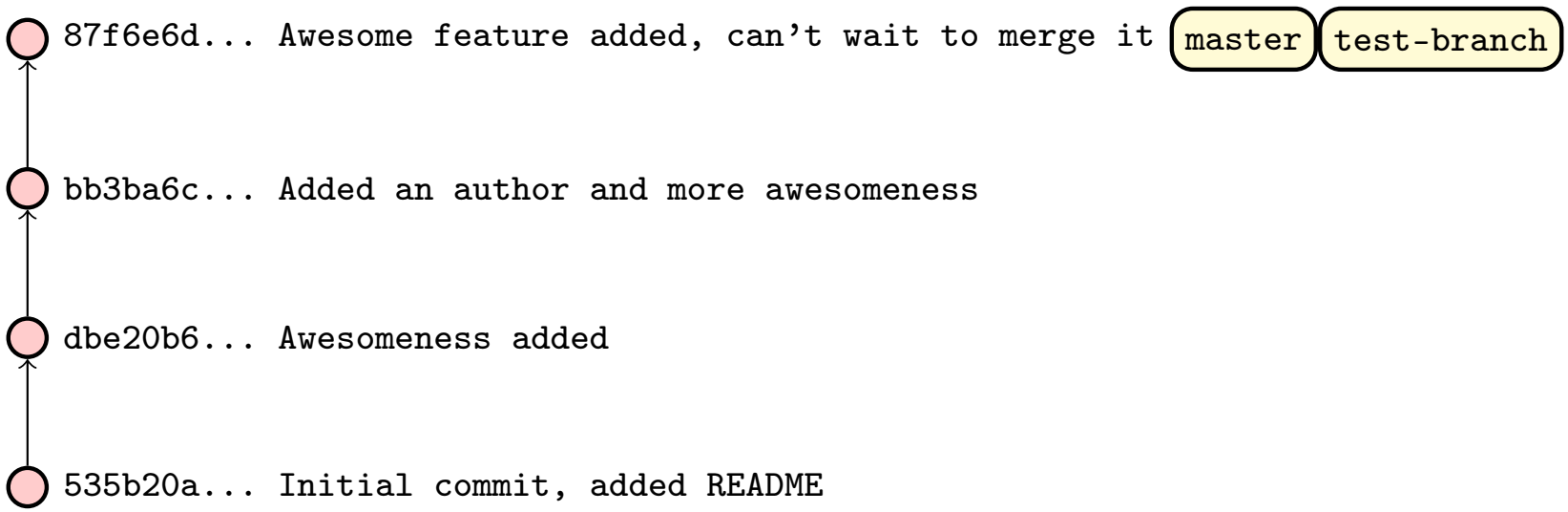
Merging

- Merging is performed by switching to the destination branch (**master** for example),
- and using the `git merge <branch-to-merge>` command,
- modification of the **branch-to-merge** branch are then applied to the current branch. If no there is no conflict, you are done.

Fast forward merge

- A fast-forward merge is made when the original branch has not been modified since the creation of the branch to merge. The pointer is just set to the merged branched SHA-1,
- the result is an history where no trace of the use of a branch to develop the feature is seen.

```
$ git checkout master

$ git merge test-branch
Updating bb3ba6c..87f6e6d
Fast-forward
 README |    1 +
 1 file changed, 1 insertion(+)
```


87f6e6d... Awesome feature added, can't wait to merge it
master
test-branch
bb3ba6c... Added an author and more awesomeness
dbe20b6... Awesomeness added
535b20a... Initial commit, added README

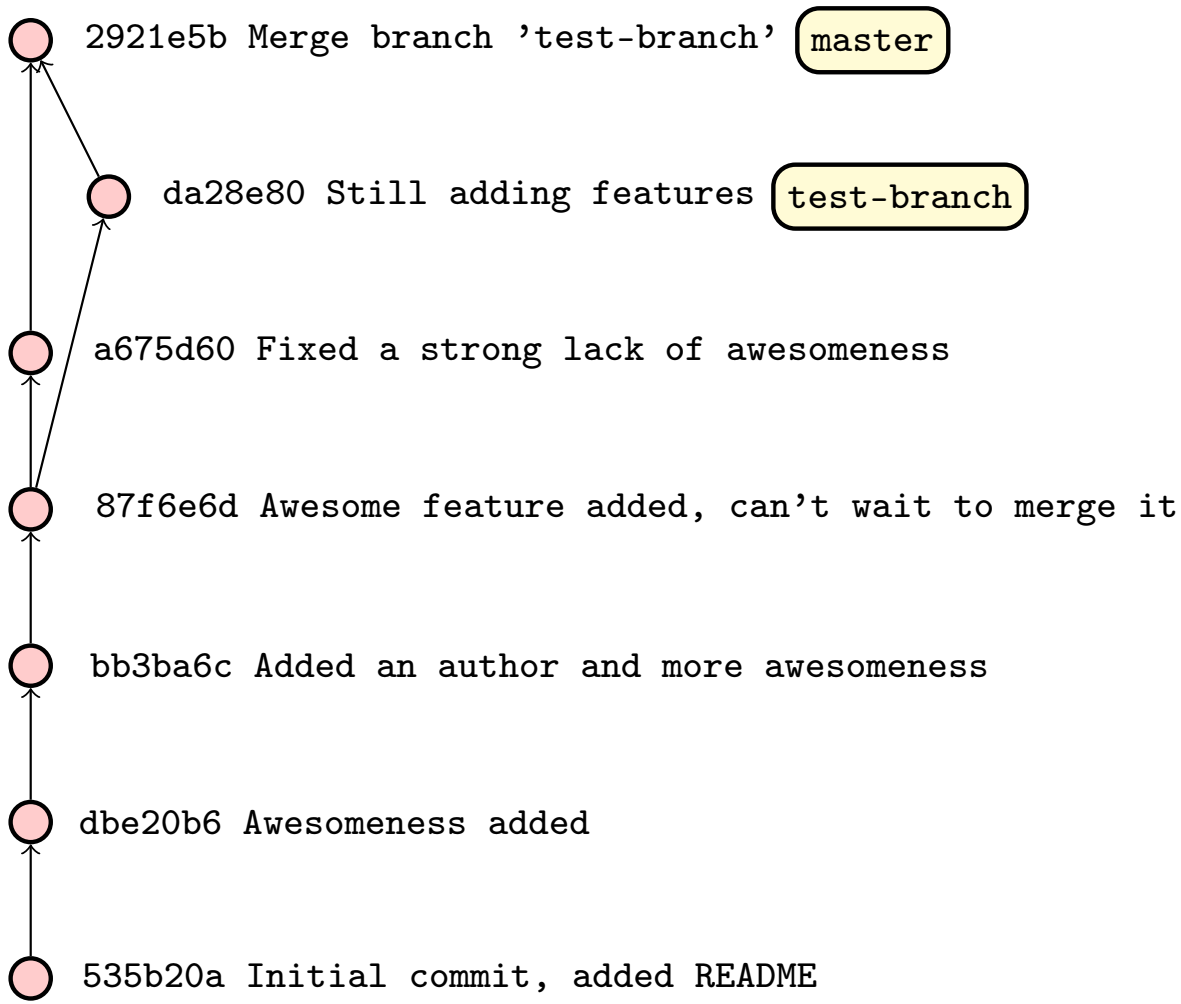
Standard merge

- When two branches have evolved their own way, a fast forward merge can not be done,
- when merging, git will automatically find the best common ancestor of both branches,
- and create a new snapshot based on the merge as well as a commit object with two parents. This object is usually referred to as a merge commit object.

```
$ git checkout test-branch

$ echo 'Features, features, features...' >> README

$ git commit -a
[test-branch da28e80] Still adding features
 1 file changed, 1 insertion(+)

$ git checkout master

$ echo 'Supercommit <super.commit@awesome.org>' >> AUTHOR

$ git commit -a
[master a675d60] Fixed a strong lack of awesomeness
 1 file changed, 1 insertion(+)
```

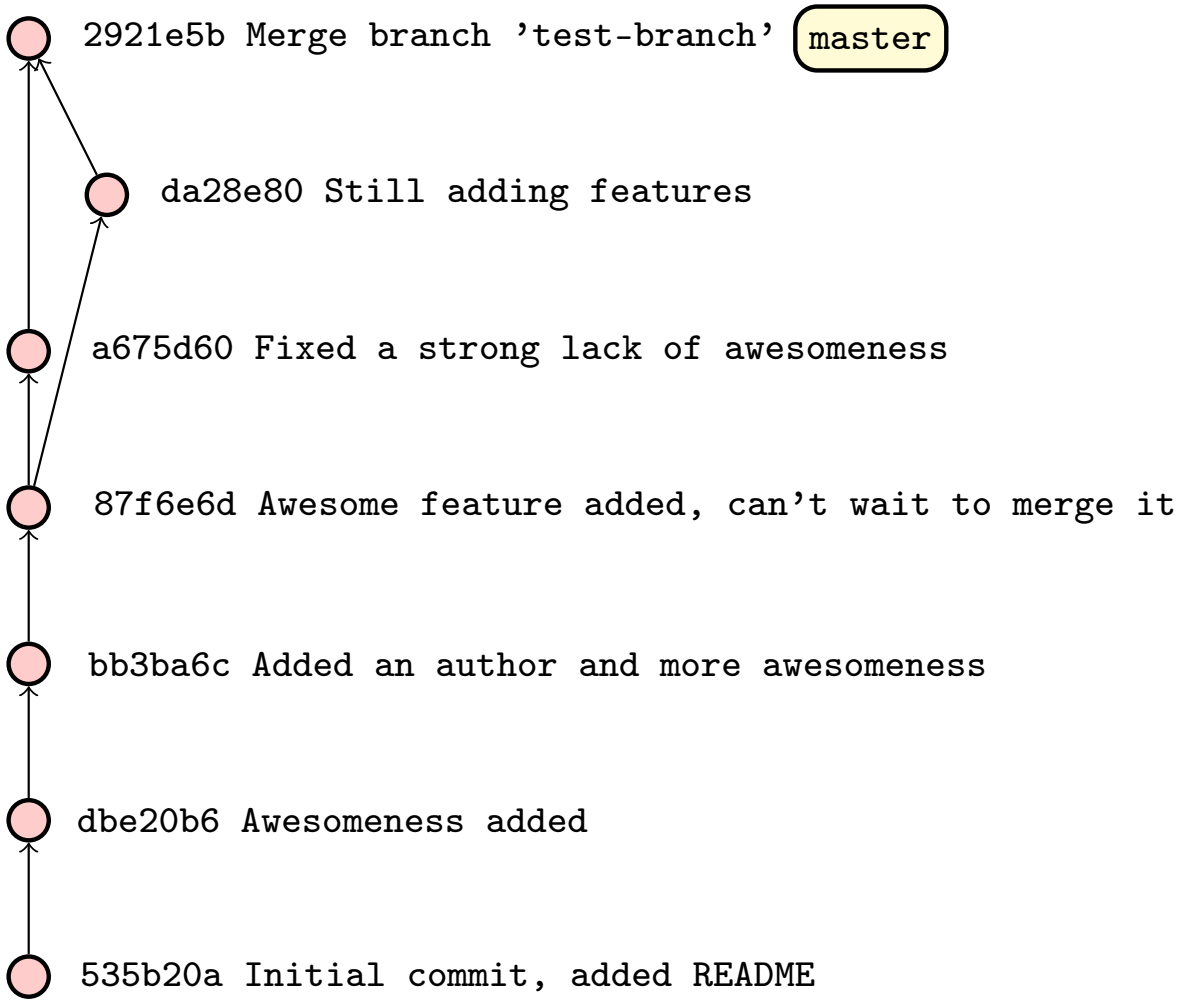

a675d60... Fixed a strong lack of awesomeness
master
da28e80... Still adding features
test-branch
87f6e6d... Awesome feature added, can't wait to merge it
bb3ba6c... Added an author and more awesomeness
dbe20b6... Awesomeness added
535b20a... Initial commit, added README

```
$ git checkout master # if not already in the master branch

$ git merge test-branch
Merge made by the 'recursive' strategy.
 README |    1 +
 1 file changed, 1 insertion(+)
```


2921e5b Merge branch 'test-branch'
master
da28e80 Still adding features
test-branch
a675d60 Fixed a strong lack of awesomeness
87f6e6d Awesome feature added, can't wait to merge it
bb3ba6c Added an author and more awesomeness
dbe20b6 Awesomeness added
535b20a Initial commit, added README

When the feature is finished and merged, the feature branch can be removed

```
$ git branch -d test-branch
Deleted branch test-branch (was da28e80).
```



Merge conflicts

- When merging two branches, a conflit occurs if a line has been modified in both branches,
- you must choose which version is good.

```
$ git checkout -b great-fix
Switched to a new branch 'great-fix'

$ echo 'Great!' >> README

$ git commit -a
[great-fix dbbe882] Great commit
 1 file changed, 1 insertion(+)

$ git checkout master
Switched to branch 'master'

$ echo 'AWE-SOME!' >> README

$ git commit -a
[master 0f1d937] Awesome commit
 1 file changed, 1 insertion(+)

$ git merge great-fix
Auto-merging README
CONFLICT (content): Merge conflict in README
Automatic merge failed; fix conflicts and then commit the result.
```

Conflict status

```
$ git status
# On branch master
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate to mark resolution)
#
# both modified:      README
#
no changes added to commit (use "git add" and/or "git commit -a")
```

- `README` file has to be manually fixed,
- the conflict is marked as solved using `git add`

Fixing conflict

```
$ cat README
[...]
Features, features, features...
<<<<<<< HEAD
AWE-SOME!
=======
Great!
>>>>>>> great-fix
```

- The current branch version is between `<<<<<<< HEAD` and `=======`
- The version of the merged branch is between `=======` and `>>>>>>> <branch-name>`
- tools such as `emacs` or `meld` can be used by calling `git mergetool`
- the tool used is set by `git config [--global] merge.tool <tool>`

Fixing conflict

```
$ vi README
$ cat README
[...]
Features, features, features...
AWE-SOME! and Great!
$ git add README
$ git commit
[master dc4cd85] Merge branch 'great-fix'
```

- When commiting, the commit message is predefined with a merge message:

  ```
  Merge branch 'great-fix'

  Conflicts:
          README
  ```

- usually, it is good to keep the `Merge branch '<branch-name>'` part and add a message describing how the conflict has been fixed.

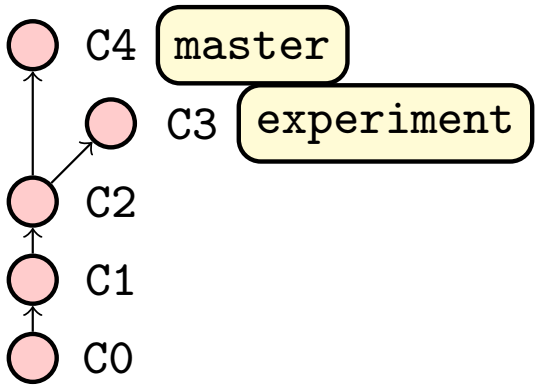

dc4cd85 Merge branch 'great-fix'
master
dbbe882 Great commit
great-fix
0f1d937 Awesome commit
2921e5b Merge branch 'test-branch'
da28e80 Still adding features
a675d60 Fixed a strong lack of awesomeness
87f6e6d Awesome feature added, can't wait to merge it
bb3ba6c Added an author and more awesomeness

Rebasing

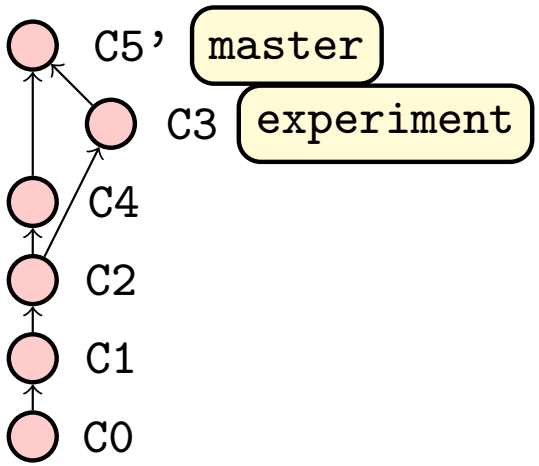
In git, you have to options for integrating changes from a branch to another

- Merging, which we already covered,
- Rebasing, which allows you to re-write history!

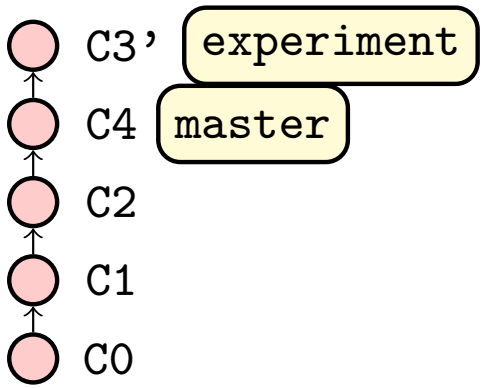
Example: initial state



With standard merge



- Rebasing is a way to pretend that you created the branch `experiment` at commit C4 instead of C2,
- If that was the case, the history would be:



- and the merge can be made using fast-forward!

Simple rebase

1. `git checkout experiment`
2. `git rebase master`

```
$ git checkout experiment
Switched to branch 'experiment'

$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: C3
```

How it works?

`git rebase [--onto <newbase>] <upstream> [<branch>]`

1. If `<branch>` is specified, first checkout that branch, otherwise work on the current branch,
2. all changes from `<upstream>` to `HEAD` are saved,
3. current branch is reset to `<upstream>` (or `<newbase>` if given),
4. the saved commits are applied **one by one** on the branch,
5. if a commit conflicts, you have to fix the conflict and use `git rebase --continue` to continue rebasing with remaining commits.

Undo!

Sometimes you want to fix a mistake you made!

- modify a commit,
- unstage files,
- undo your local changes to a file.

Modify commit

- `git commit --amend`
  - takes your staging area and add it to the last made commit,
  - if no changes where made, just edit the commit message
  - warning! if you change anything (including the commit message, author...), the SHA-1 will be changed as well. Do NOT modify a commit that has already been pushed to a remote[1]!
  - can be used to change the author (`--author=`), date (`--date=`)...

[1] more on remotes later

Unstaging file

You can remove files from the staging area

- You made a wrong `git add` of a file that should not go to the next commit
- you can unstage the file using `git reset` or a part of a file `git reset -p`

```
$ vi README
$ git status
# Changes not staged for commit:
# modified:   README

$ git add README

$ git status
# Changes to be committed:
# modified:   README

$ git reset -- README
Unstaged changes after reset:
M README

$ git status
# Changes not staged for commit:
# modified:   README
```

Undo local changes

Reset all to a given commit

- `git reset --hard`
- resets the state of the index and the working dir,
- all local changes are lost,
- can not be used for a single file!

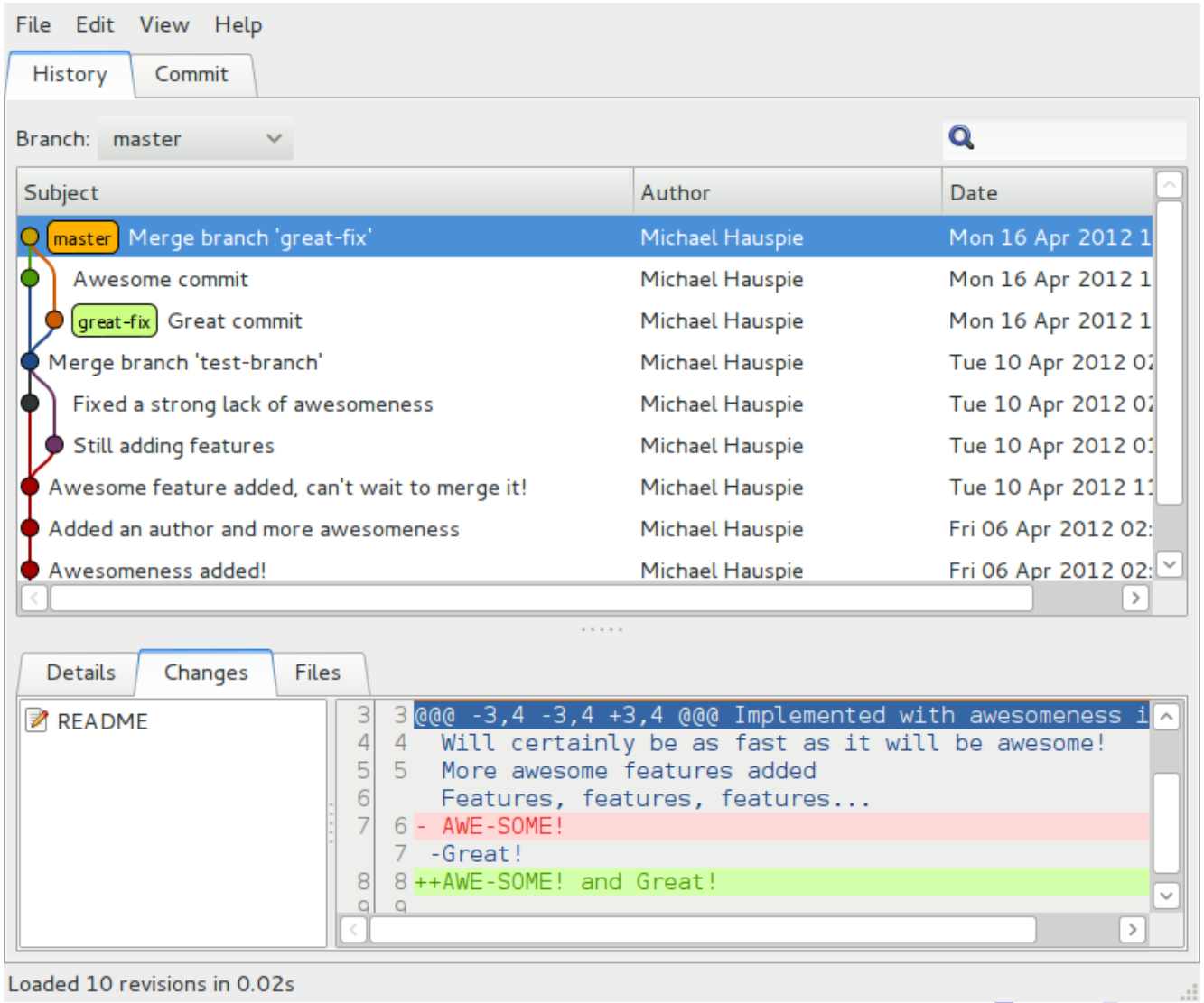
Undo changes to a particular file

- `git checkout -- <path>`
- only changes the file(s) in the working directory,
- if you had already staged something, you have to unstage using `git reset`

Visual tools: gitk, gitg

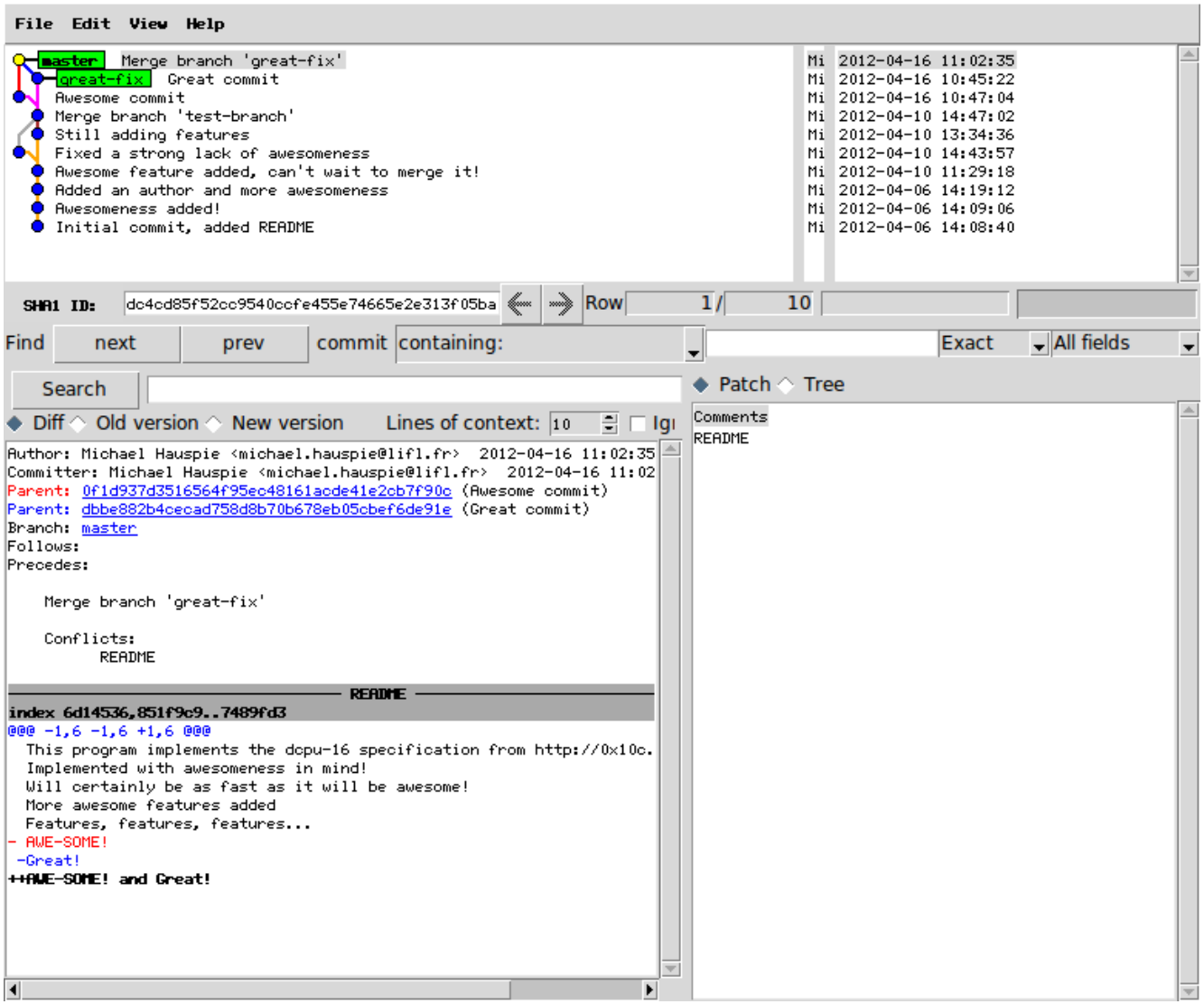
Gitg

- Visual tool that integrates with gnome



Gitk

- Tcl/Tk Visual tool



Using remotes

Cloning an existing repository

- The easiest way to work with a remote is to clone an existing repository,
- `git clone <url>`,
- common url forms:
  - `path/to/some/repository` → local folder
  - `login@server:path/to/some/repository` → ssh
  - `git+ssh://login@server:port/path/to/some/repository` → ssh
  - `https://login@server:port/path/to/some/repository` → https
  - `git://server/path/to/some/repository` → anonymous git protocol (read only)

```
$ git clone git@github.com:hauspie/rflpc
Cloning into 'rflpc'...
remote: Counting objects: 2412, done.
remote: Compressing objects: 100% (849/849), done.
remote: Total 2412 (delta 1529), reused 2397 (delta 1514)
Receiving objects: 100% (2412/2412), 6.63 MiB | 2.17 MiB/s, done.
Resolving deltas: 100% (1529/1529), done.
```

What happened?

1. A folder is created and initialized,
2. a remote is added,
3. fetched (*i.e.* objects from the remote are downloaded),
4. and the working directory is set to the commit referenced by HEAD (usually the master branch[2]).

[2] the only way to change it is to be able to change the `.git/HEAD` file on the remote side

Listing remotes

Remotes are managed using `git remote`

```
$ git remote
origin

$ git remote -v
origin git@github.com:hauspie/rflpc (fetch)
origin git@github.com:hauspie/rflpc (push)
```

What is a remote?

A remote is

- a set of **read only** branches `remotes/remote-name/<branches-name>`
- an url to fetch objects from (download objects),
- an url to push objects to (upload objects),

Listing remotes and local branches

- `git branch -a`

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/develop
  remotes/origin/feature/i2c-driver
  remotes/origin/gh-pages
  remotes/origin/master
```

Defining a remote

- A remote can be added using `git remote add <remote-name> <url>`

```
$ git remote add francois git://github.com/fser/rflpc

$ git remote -v
francois git://github.com/fser/rflpc (fetch)
francois git://github.com/fser/rflpc (push)
origin git@github.com:hauspie/rflpc (fetch)
origin git@github.com:hauspie/rflpc (push)
```

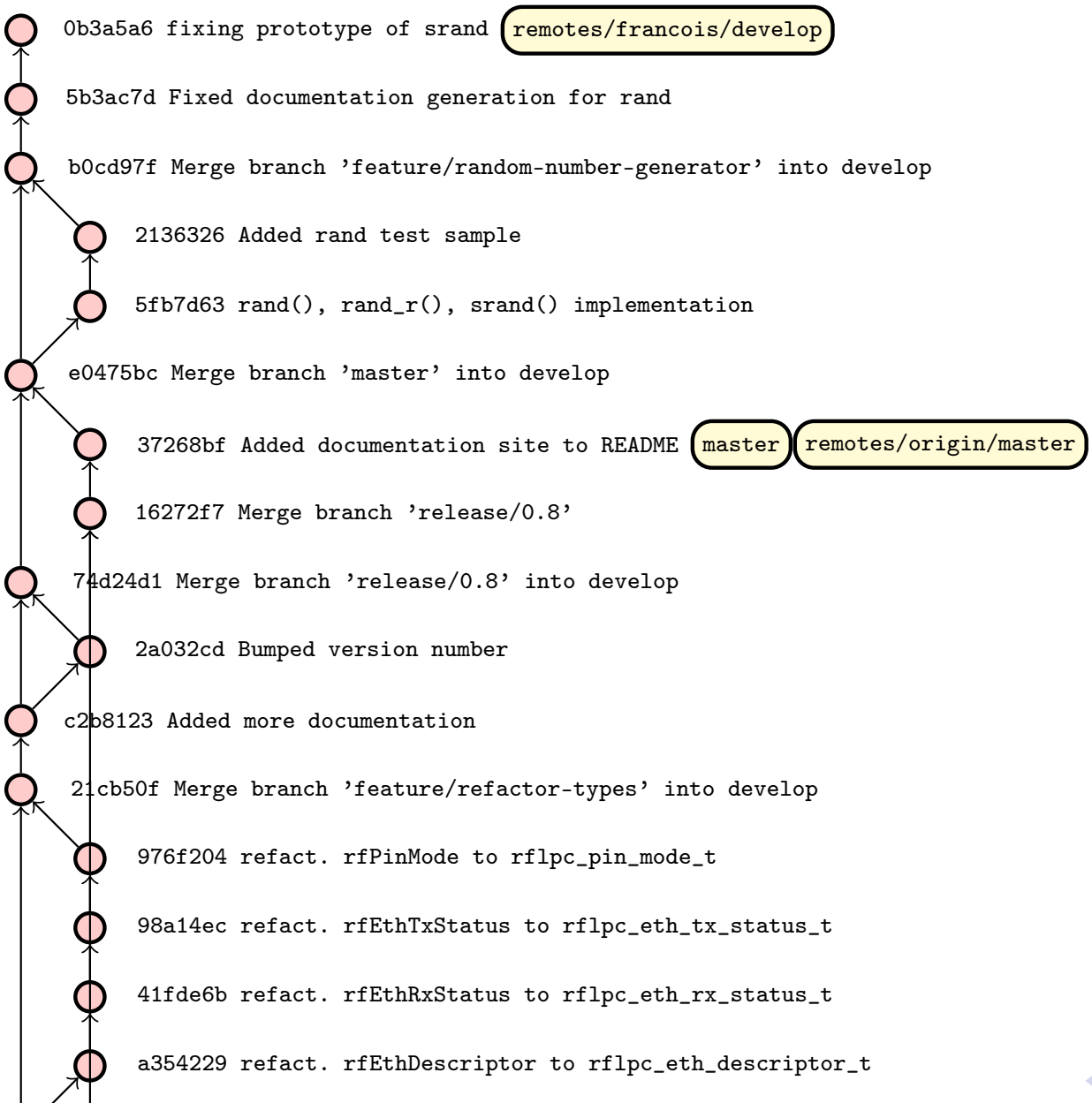
- Adding a remote does not download objects. You have to fetch!

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/develop
  remotes/origin/feature/i2c-driver
  remotes/origin/gh-pages
  remotes/origin/master
$ git fetch francois
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (5/5), done.
Unpacking objects: 100% (9/9), done.
remote: Total 9 (delta 4), reused 8 (delta 4)
From git://github.com/fser/rflpc
 * [new branch]      develop    -> francois/develop
 * [new branch]      gh-pages   -> francois/gh-pages
 * [new branch]      master     -> francois/master
```

Check branch state

`git branch -av` will show the SHA-1 of all branches, as well as the short commit message

```
$ git branch -av
* master                                37268bf Added documentation site to README
  remotes/francois/develop              0b3a5a6 fixing prototype of srand
  remotes/francois/gh-pages             31f027c Fixed main site page
  remotes/francois/master               c586a17 Merge branch 'master' of https://github.com/hauspie/
  remotes/origin/HEAD                   -> origin/master
  remotes/origin/develop                a389e14 Merge branch 'feature/read-unique-identifier' into d
  remotes/origin/feature/i2c-driver     a2ff0b4 Init function should properly init clock and pins
  remotes/origin/gh-pages               31f027c Fixed main site page
  remotes/origin/master                 37268bf Added documentation site to README
```


0b3a5a6 fixing prototype of srand
remotes/francois/develop
5b3ac7d Fixed documentation generation for rand
b0cd97f Merge branch 'feature/random-number-generator' into develop
2136326 Added rand test sample
5fb7d63 rand(), rand_r(), srand() implementation
e0475bc Merge branch 'master' into develop
37268bf Added documentation site to README
master
remotes/origin/master
16272f7 Merge branch 'release/0.8'
74d24d1 Merge branch 'release/0.8' into develop
2a032cd Bumped version number
c2b8123 Added more documentation
21cb50f Merge branch 'feature/refactor-types' into develop
976f204 refact. rfPinMode to rflpc_pin_mode_t
98a14ec refact. rfEthTxStatus to rflpc_eth_tx_status_t
41fde6b refact. rfEthRxStatus to rflpc_eth_rx_status_t
a354229 refact. rfEthDescriptor to rflpc_eth_descriptor_t

Local branch to track remote branch

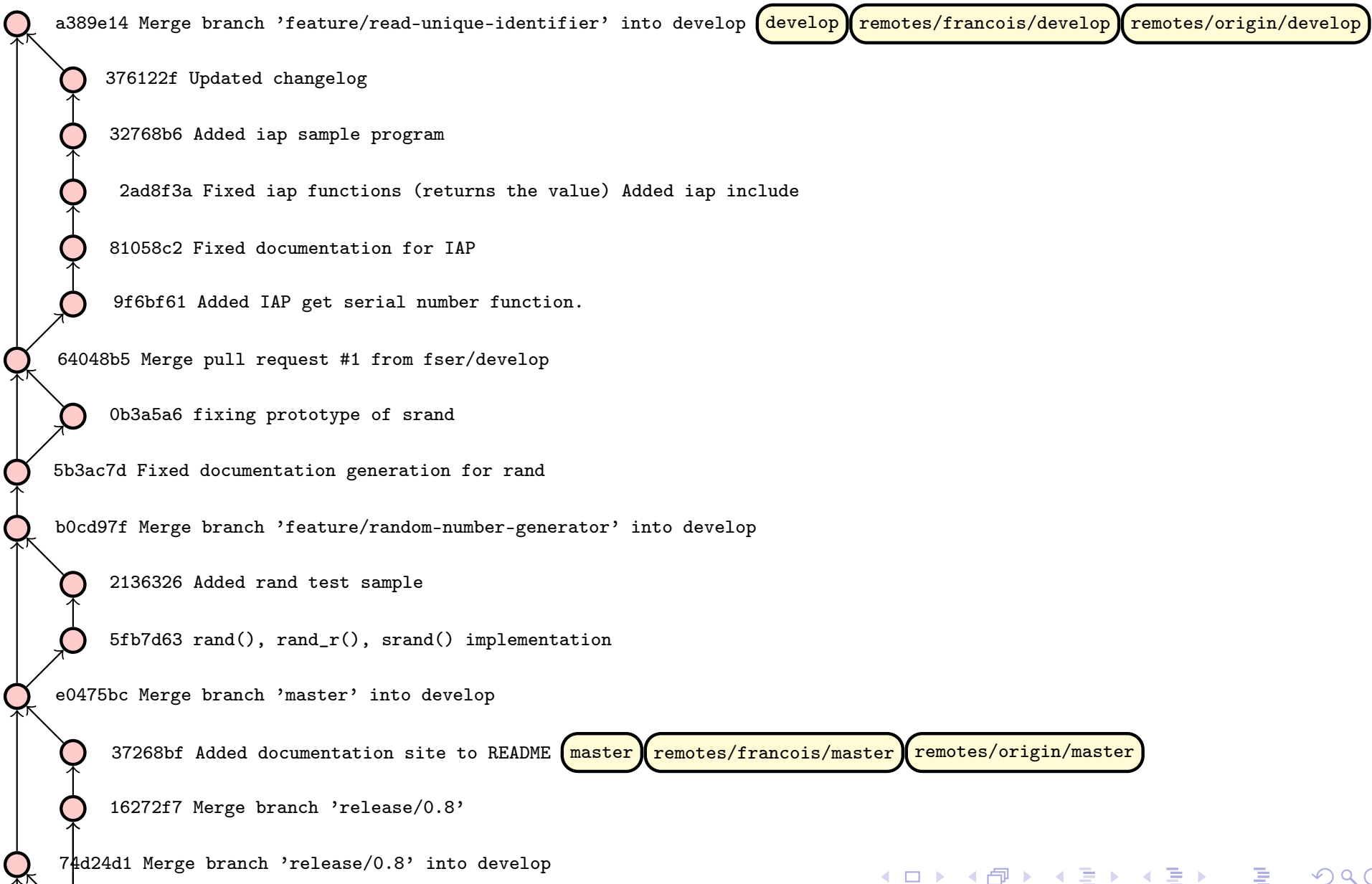
- You can checkout a remote branch

```
$ git checkout francois/develop
Note: checking out 'francois/develop'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 0b3a5a6... fixing prototype of srand
```

- you can commit, but there is no easy reference to your commits (only the SHA-1),
- the remote branch is a **constant** pointer to a commit object and thus can not be updated locally,
- You have to create another pointer (*i.e.* branch) that points to the same commit object.

- A branch can be created pointing to any object with `git branch <branch-name> <commit-ref>` or `git checkout -b <branch-name> <commit-ref>`,
- where <commit-ref> is a SHA-1, a branch name, tag name...

```
$ git checkout -b francois/develop francois/develop
Branch francois/develop set up to track remote branch develop from francois.
Switched to a new branch 'francois/develop'

$ git branch -av
* francois/develop                a389e14 Merge branch 'feature/read-unique-identifier'
  master                          37268bf Added documentation site to README
  remotes/francois/develop        a389e14 Merge branch 'feature/read-unique-identifier'
  remotes/francois/gh-pages       31f027c Fixed main site page
  remotes/francois/master         37268bf Added documentation site to README
  remotes/francois/testmickey     37268bf Added documentation site to README
  remotes/origin/develop          a389e14 Merge branch 'feature/read-unique-identifier'
  remotes/origin/gh-pages         31f027c Fixed main site page
  remotes/origin/master           37268bf Added documentation site to README
```


a389e14 Merge branch 'feature/read-unique-identifier' into develop
develop
remotes/francois/develop
remotes/origin/develop
376122f Updated changelog
32768b6 Added iap sample program
2ad8f3a Fixed iap functions (returns the value) Added iap include
81058c2 Fixed documentation for IAP
9f6bf61 Added IAP get serial number function.
64048b5 Merge pull request #1 from fser/develop
0b3a5a6 fixing prototype of srand
5b3ac7d Fixed documentation generation for rand
b0cd97f Merge branch 'feature/random-number-generator' into develop
2136326 Added rand test sample
5fb7d63 rand(), rand_r(), srand() implementation
e0475bc Merge branch 'master' into develop
37268bf Added documentation site to README
master
remotes/francois/master
remotes/origin/master
16272f7 Merge branch 'release/0.8'
74d24d1 Merge branch 'release/0.8' into develop

Tracking a remote with a already existing local branch

- If you already have a local branch that does **not yet track** the remote,
- `git branch --set-upstream <local-branch> <remote-branch>`

Update your local branch with remote changes

Fetching the remote

- `git fetch <remote>` fetches (downloads) objects from a remote,
  - it does **not** modify your working directory neither your local branch

Merging the changes

- Merging a remote branch is **the same operation** as merging another local branch:
  1. Switch to branch your want to merge **into**,
  2. issue a `git merge remote/remote_branch`.

Doing it faster

- If your local branch **tracks** a remote branch then
  - `git pull` automatically does the **fetch** and the **merge** operation
  - `git pull <remote>` and `git pull <remote> <branch-name>` also works.

Pushing objects to the remote

Sending your work to the remote is made using the

`git push <remote> [<local-ref>][:<destination-ref>]`

- pushes the objects referenced by `<local-ref>` to the remote and create/update the reference named `<destination-ref>` on the remote,
- if `:<destination-ref>` is omitted, the path used for the destination reference is the same as the local one,
- if `<local-ref>` is omitted, this action deletes `<destination-ref>`,
- if both `<local-ref>` and `<destination-ref>` are omitted (but the `:` kept), all local branches matching a remote branch are pushed

Examples

1. `git push`, pushes the current branch its current remote,
2. `git push origin :`, pushes all matching branches to `origin`,
3. `git push origin`. By default, same behavior as the previous command, but can be reconfigured,
4. `git push origin master`, pushes the master branch to `origin`
5. `git push origin HEAD`, pushes the current branch to a branch with the same name in `origin`,
6. `git push origin new-feature:hauspie-new-feature`, pushes the branch `new-feature` to a branch named `hauspie-new-feature` in `origin`
7. `git push origin :experimental`, deletes the `experimental` branch in origin.

Last warning

Never change a commit object (by amending, rebasing...) if it has already been pushed to a remote !!

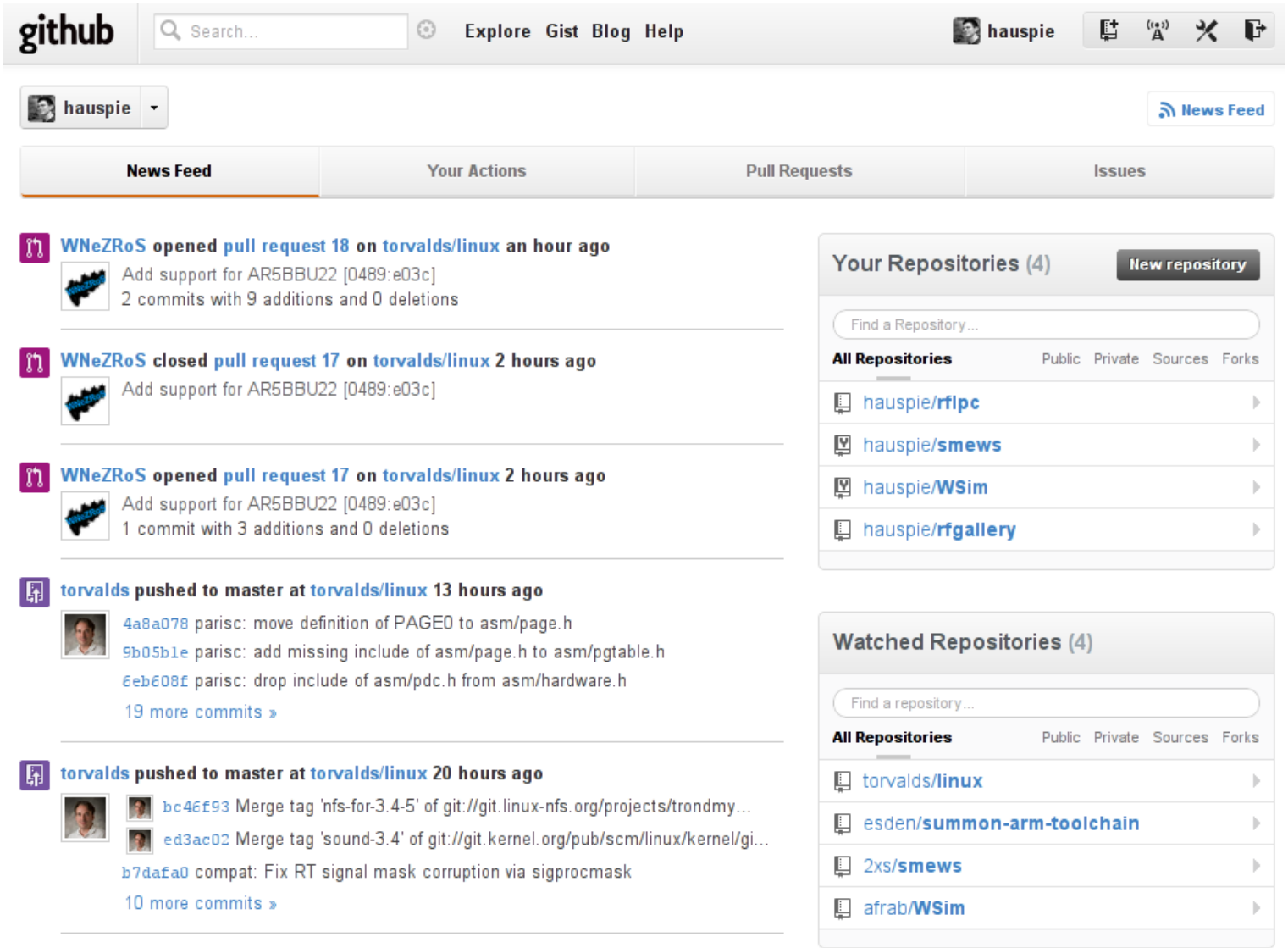
Web tools

Web tools

Popular git based web tools

- Github
  - Reference site for git based open source software
  - Great interface, easy to use
  - No private projects (for free)
- Gitlab
  - The tool itself is open source and can be hosted internaly
  - Can host private projects

Github


github
Search...
Explore Gist Blog Help
hauspie
hauspie
News Feed
News Feed
Your Actions
Pull Requests
Issues
WNeZRoS opened pull request 18 on torvalds/linux an hour ago
Add support for AR5BBU22 [0489:e03c]
2 commits with 9 additions and 0 deletions
WNeZRoS closed pull request 17 on torvalds/linux 2 hours ago
Add support for AR5BBU22 [0489:e03c]
WNeZRoS opened pull request 17 on torvalds/linux 2 hours ago
Add support for AR5BBU22 [0489:e03c]
1 commit with 3 additions and 0 deletions
torvalds pushed to master at torvalds/linux 13 hours ago
4a8a078 parisc: move definition of PAGE0 to asm/page.h
9b05b1e parisc: add missing include of asm/page.h to asm/pgtable.h
6eb608f parisc: drop include of asm/pdc.h from asm/hardware.h
19 more commits »
torvalds pushed to master at torvalds/linux 20 hours ago
bc46f93 Merge tag 'nfs-for-3.4-5' of git://git.linux-nfs.org/projects/trondmy...
ed3ac02 Merge tag 'sound-3.4' of git://git.kernel.org/pub/scm/linux/kernel/gi...
b7dafa0 compat: Fix RT signal mask corruption via sigprocmask
10 more commits »
Your Repositories (4)
New repository
Find a Repository...
All Repositories Public Private Sources Forks
hauspie/rflpc
hauspie/smews
hauspie/WSim
hauspie/rfgallery
Watched Repositories (4)
Find a repository...
All Repositories Public Private Sources Forks
torvalds/linux
esden/summon-arm-toolchain
2xs/smews
afrab/WSim

Forks

When you want to collaborate on a project hosted on github the best way to do so is to:

1. fork the project,
2. clone your forked version,
3. add a remote pointing to the "real" (or upstream) project (to keep track of the work of the original contributors),
4. work on your version of the project (change, commit, push...),
5. Issue a pull request when you want your changes to be merged in the main repository.

Working with your forked project

Suppose there is the corresponding setup:

- The upstream project's url is `git://github.com/2xs/smews.git`,
- My fork's url is `git@github.com:hauspie/smews.git`,

Working with your forked project

Suppose there is the corresponding setup:

- The upstream project's url is `git://github.com/2xs/smews.git`,
- My fork's url is `git@github.com:hauspie/smews.git`,

1. `git clone git@github.com:hauspie/smews.git`

Working with your forked project

Suppose there is the corresponding setup:

- The upstream project's url is `git://github.com/2xs/smews.git`,
- My fork's url is `git@github.com:hauspie/smews.git`,

1. `git clone git@github.com:hauspie/smews.git`
2. `git remote add upstream git://github.com/2xs/smews.git`

Working with your forked project

Suppose there is the corresponding setup:

- The upstream project's url is `git://github.com/2xs/smews.git`,
- My fork's url is `git@github.com:hauspie/smews.git`,

1. `git clone git@github.com:hauspie/smews.git`
2. `git remote add upstream git://github.com/2xs/smews.git`
3. `git fetch upstream`

Working with your forked project

Suppose there is the corresponding setup:

- The upstream project's url is `git://github.com/2xs/smews.git`,
- My fork's url is `git@github.com:hauspie/smews.git`,

1. `git clone git@github.com:hauspie/smews.git`
2. `git remote add upstream git://github.com/2xs/smews.git`
3. `git fetch upstream`
4. `git branch -av`

```
* master                      670cb5b Merge branch 'hotfix/1.6.2c'
  remotes/upstream/develop         92e55b8 Merge branch 'feature/use-rflpc-0.7' into develop
  remotes/upstream/feature/tls     3b5106b repos reorg
  remotes/upstream/master          670cb5b Merge branch 'hotfix/1.6.2c'
  remotes/origin/HEAD          -> origin/master
  remotes/origin/develop       92e55b8 Merge branch 'feature/use-rflpc-0.7' into develop
  remotes/origin/feature/tls 3b5106b repos reorg
  remotes/origin/master        670cb5b Merge branch 'hotfix/1.6.2c'
```

- Updating your version is then just a matter of merging branches.

Pull requests

Pull request is a way to ask a project maintainer to integrate your modification to the upstream.

- code review can be done using github (and must be!),
- pull requests can be amended (to reflect comments in the code review for example),
- pull request can be merged either by the web interface (if no conflict is generated by the merge) or by using *fetch and merge* or *patches*

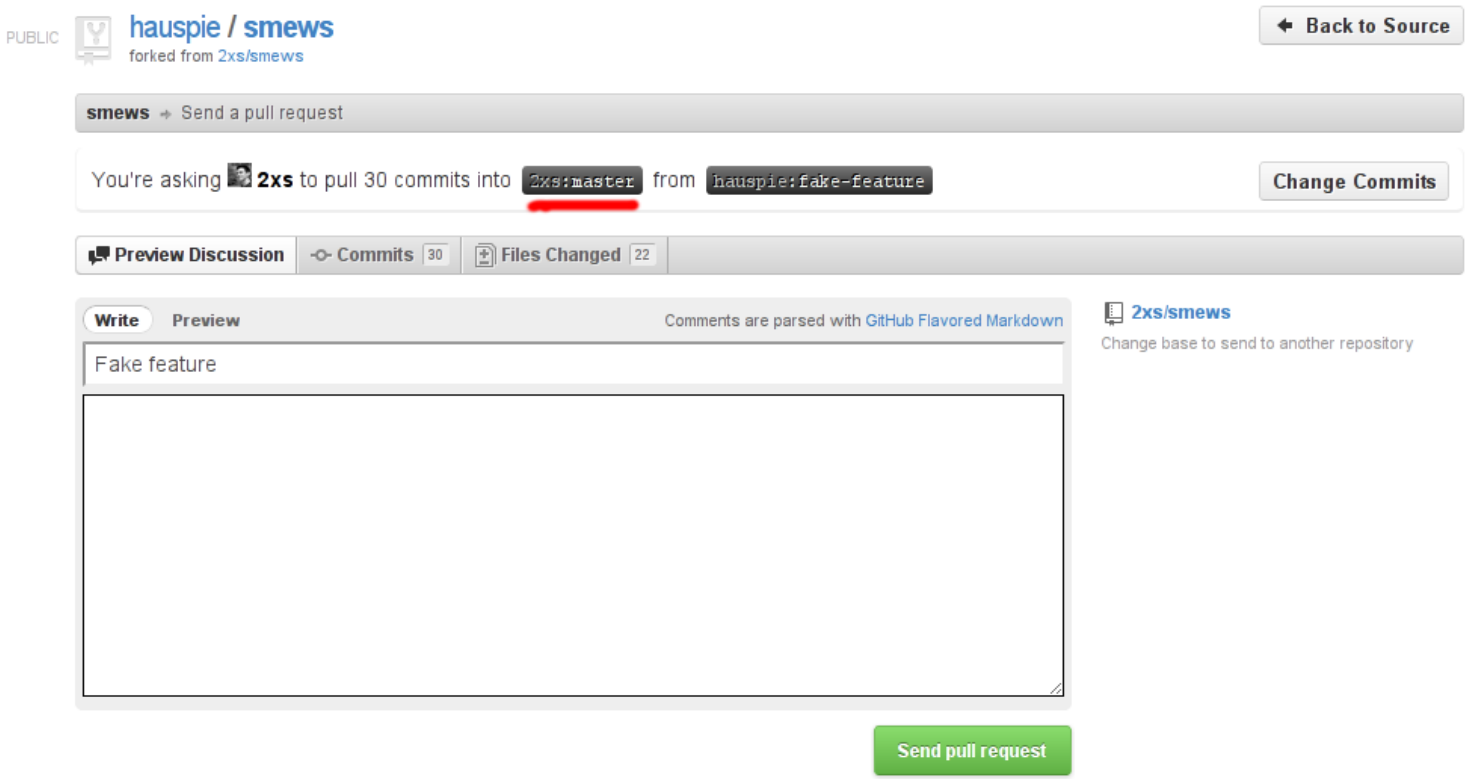
Creating a pull request

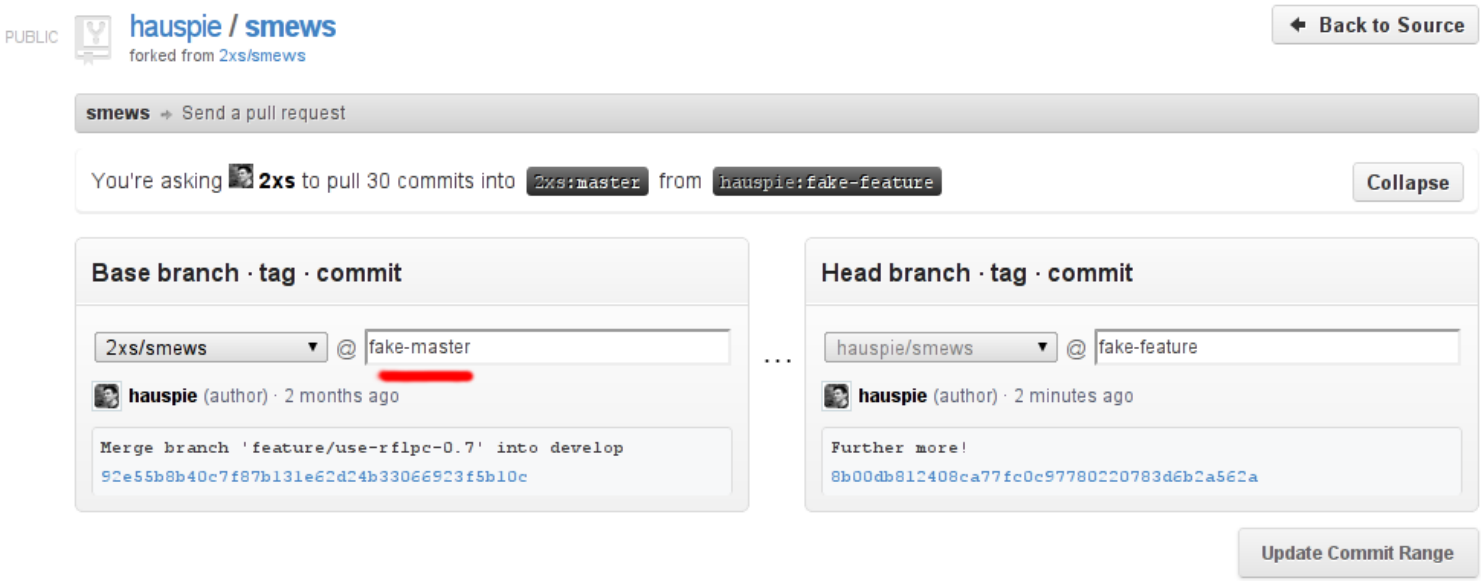
Repository situation

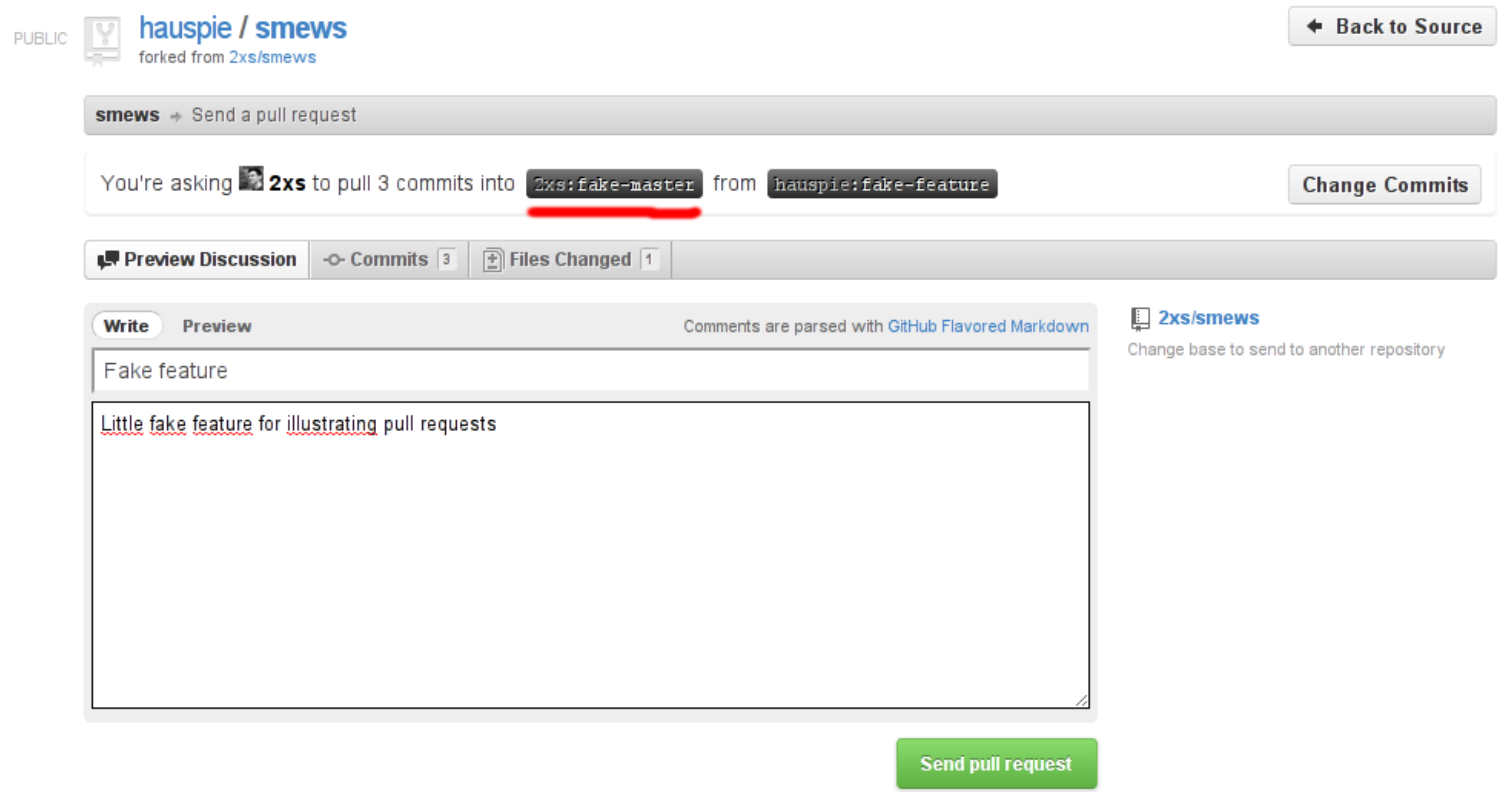
1. A feature is developed in the `fake-feature` branch,
2. The integration will be made in the `fake-master`

Making the pull request

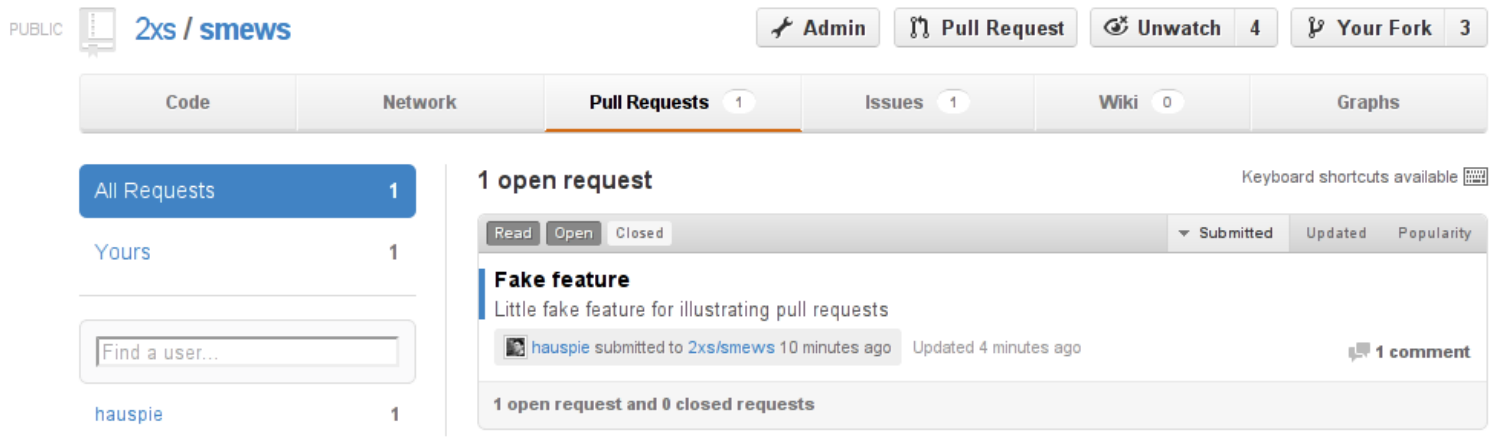
1. Switch to the `fake-feature` branch in the github website,
2. click the pull request button,
3. update the commit ranges (mainly select the destination branch),
4. add a message and send the pull request


PUBLIC
hauspie / smews
forked from 2xs/smews
Back to Source
smews → Send a pull request
You're asking 2xs to pull 30 commits into 2xs:master from hauspie:fake-feature
Change Commits
Preview Discussion
Commits 30
Files Changed 22
Write
Preview
Comments are parsed with GitHub Flavored Markdown
Fake feature
2xs/smews
Change base to send to another repository
Send pull request

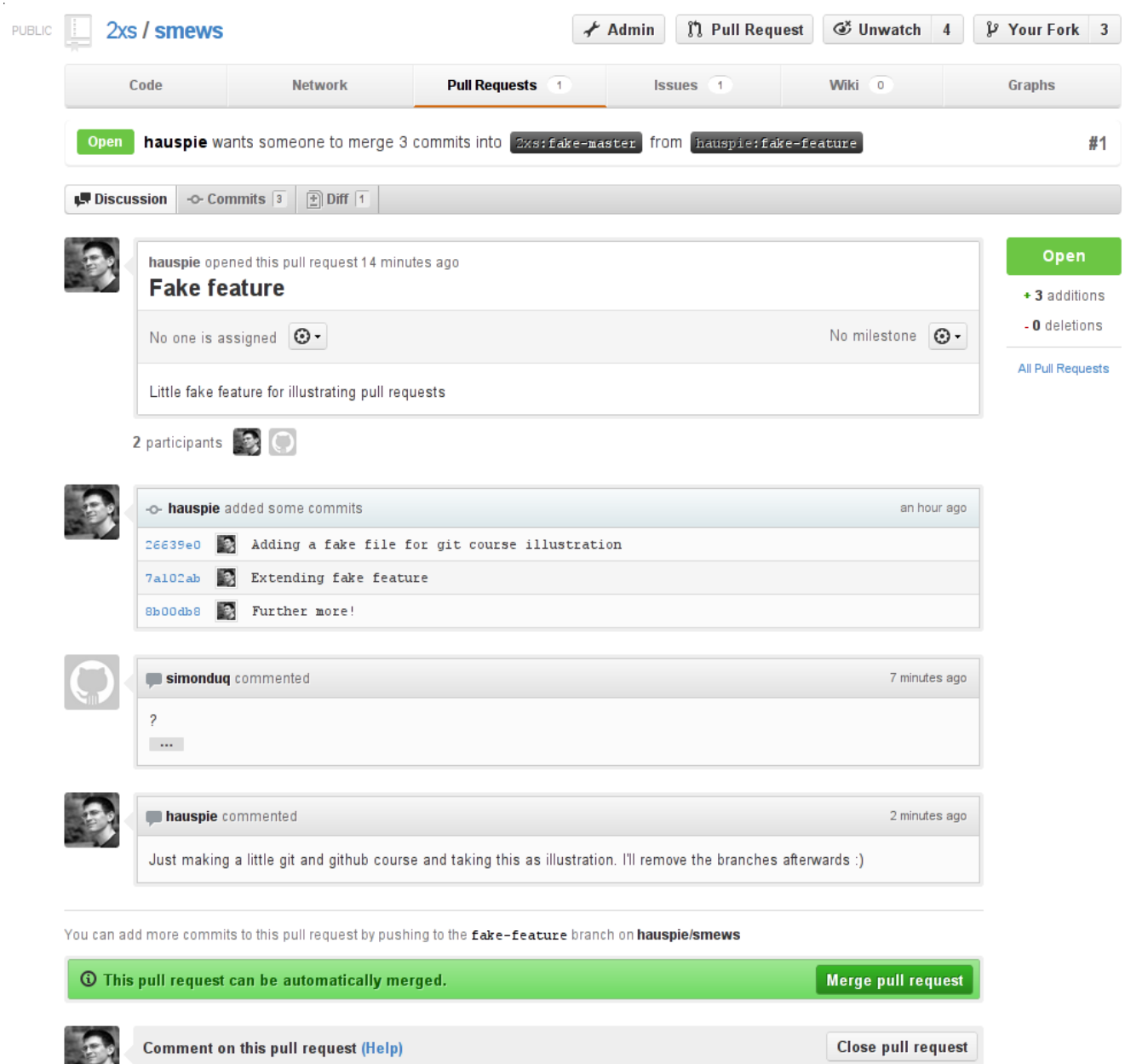

PUBLIC
hauspie / smews
forked from 2xs/smews
Back to Source
smews ← Send a pull request
You're asking 2xs to pull 30 commits into 2xs:master from hauspie:fake-feature
Collapse
Base branch · tag · commit
2xs/smews
@
fake-master
hauspie (author) · 2 months ago
Merge branch 'feature/use-rflpc-0.7' into develop
92e55b8b40c7f87b131e62d24b33066923f5b10c
...
Head branch · tag · commit
hauspie/smews
@
fake-feature
hauspie (author) · 2 minutes ago
Further more!
8b00db812408ca77fc0c97780220783d6b2a562a
Update Commit Range


PUBLIC
hauspie / smews
forked from 2xs/smews
Back to Source
smews → Send a pull request
You're asking 2xs to pull 3 commits into 2xs:fake-master from hauspie:fake-feature
Change Commits
Preview Discussion
Commits 3
Files Changed 1
Write
Preview
Comments are parsed with GitHub Flavored Markdown
Fake feature
Little fake feature for illustrating pull requests
2xs/smews
Change base to send to another repository
Send pull request

Reviewing requests



Reviewing requests



Merging the request

Three possible methods

1. If the pull does not create a conflict, github will allow to merge it by clicking a button,
2. Usual *Fetch and merge*. The maintainer adds a remote to the fork, fetches the objects, merge and push back to the upstream,
3. Use `git am` using the automatically generated patch file located at `http://github.com/user/project/pull/<#pull>.patch`

Fetch and merge

In the upstream repository:

```
$ git checkout fake-master
$ git remote add hauspie git://github.com/hauspie/smews.git
$ git fetch hauspie
$ git merge hauspie/fake-feature
$ git push origin fake-master
```

Merge with git am

If you manage lots of contributors, adding a remote each time you have to merge a pull request can be a pain.

- `git am`: Apply a series of patches from a mailbox
- this command is made to process outputs of the `git format-patch` command,
- for each pull request, github generates such a patch file suitable for `git am`.

```
$ git checkout fake-master
$ curl https://github.com/2xs/smews/pull/1.patch | git am
$ git push origin fake-master
```

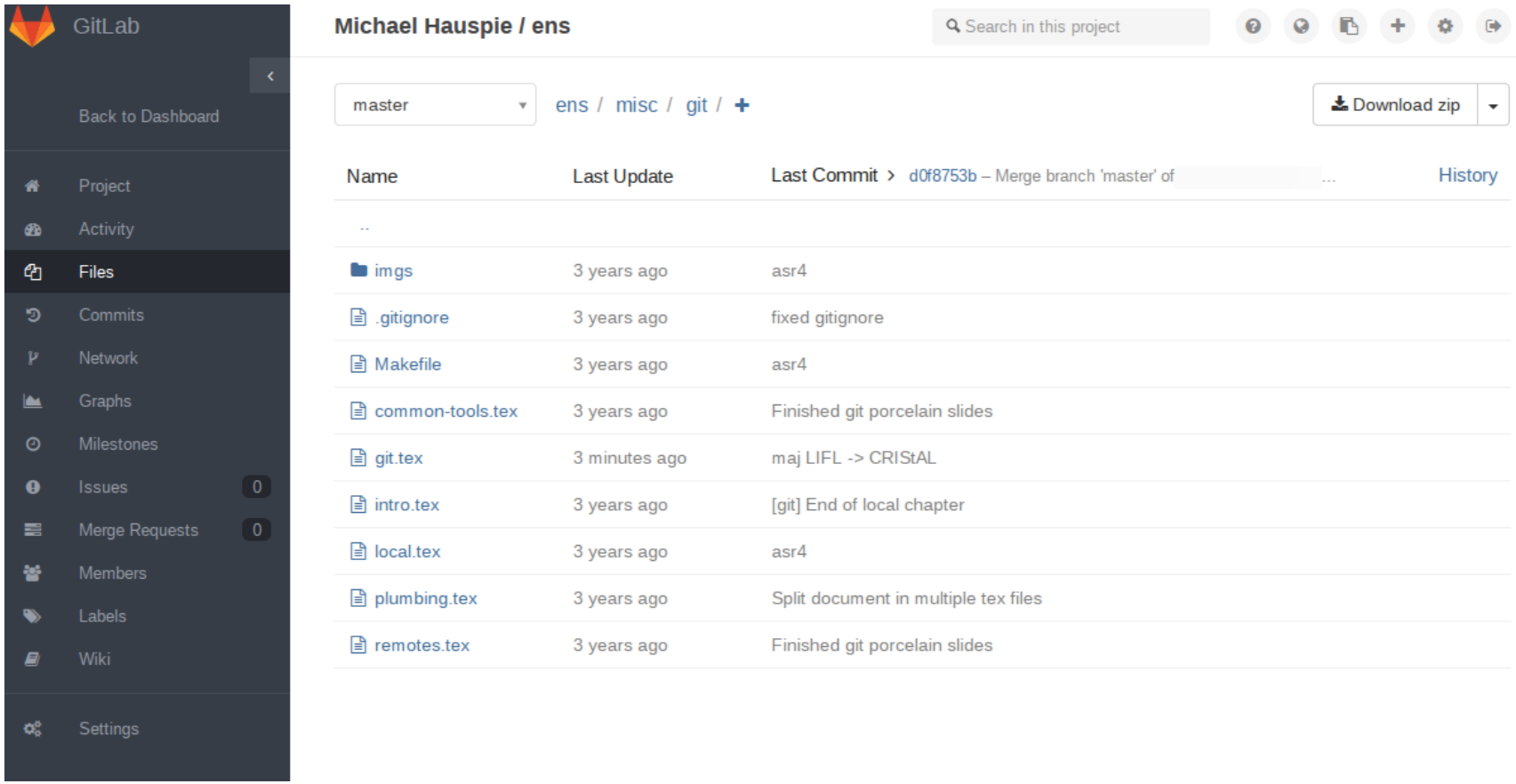
Good practice

For complicated features, it is a good idea to merge in a new branch in the upstream repository. This can prevent your master branch to be polluted while you perform a difficult merge.

```
$ git checkout -b integrate/1
$ git merge hauspie/fake-feature
$ git checkout fake-master
$ git merge integrate/1
```

Gitlab

Gitlab is tool that can be provided as a service (such as github) or that you can install locally. It is quite similar to github.



Acknowledgment and licensing

Acknowledgment and licensing

Acknowledgment

- This presentation includes materials from the Pro Git book by Scott Chacon,
- many thanks to him for this great book.

License