GIT - THERE BE DRAGONS!

from *(l)user* to *r00t* in 60 minutes

Javier L. Gómez

January 28, 2019

Computer Architecture and Technology Area (ARCOS)—University Carlos III of Madrid

LICENSE

Agenda

Introduction

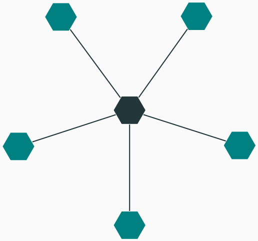
SCM/Revision control systems

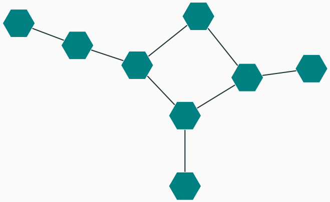
“Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later (…) if you screw things up or lose files, you can easily recover.” [`https:///git-scm.org/`]

What you get:

- Compare changes over time or revert files.
- See who introduced an issue.
- Make experimental changes (and merge them).
- …

RCS models: centralized/distributed


Centralized: Subversion (SVN), CVS...


Distributed: git, Mercurial (hg)...

git - the stupid content tracker (1/3)

This is not GitHub, nor GitLab...

git - the stupid content tracker (2/3)

Git: a distributed RCS.

Started by Linus Torvalds; currently maintained by Junio C Hamano.

git - the stupid content tracker (3/3)

- 139 separate binaries, wrapped by `git(1)`; some of them accept lots of options! e.g. `git-log` parses 100+ options
- Divided into high level (porcelain) and low level (plumbing) commands
- Largely documented:

  ```
  $ basename --suffix=.1.gz
  /usr/share/man/man1/git* | xargs man | wc -l
  53260
  ```
  (=870 pages PDF)
- Target of this talk: *people using Git*

Minimum set of commands

Initialization	`git clone`
	`git init`

Interrogation	`git log`
	`git status`
	`git diff`

Manipulation	`git add`
	`git commit`

Interaction	`git push`
	`git pull`

Git essentials

Working tree and .git/ directory (1/2)

`.git/` directory: contains Git administrative and control files.

Working tree: the tree of checked out files.

```
repository/
├── .git/
│   ├── config
│   ├── HEAD
│   └── …
├── Makefile
├── main.cpp
└── …
```

Working tree and .git/ directory (2/2)

Bare repository: NO working tree + NO `.git/` directory sub-directory.
Git files directly present in the directory.

```
repository.git/
├── config
├── HEAD
└── …
```

Objects, references and symrefs (1/3)

Object: raw octets stored in Git; identified by its SHA-1.
Types: *commit*, *tree*, *blob*, *tag*.

Objects, references and symrefs (1/3)

Object: raw octets stored in Git; identified by its SHA-1.
Types: *commit*, *tree*, *blob*, *tag*.

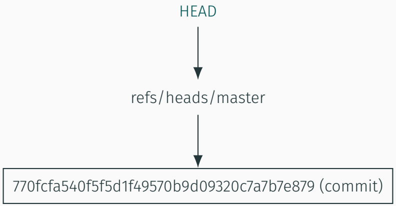
Ref(erence): a name that points to an object. Hierarchical namespace rooted at `refs/`

Objects, references and symrefs (1/3)

Object: raw octets stored in Git; identified by its SHA-1.
Types: *commit*, *tree*, *blob*, *tag*.

Ref(erence): a name that points to an object. Hierarchical namespace rooted at `refs/`

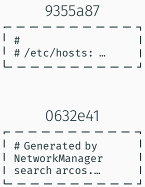
Symref: a ref that points to another ref, e.g. HEAD.



Objects, references and symrefs (2/3)

The contents of an object depend on its type:

Blob: raw data; stores file contents.

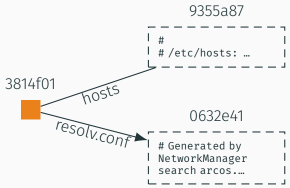


Objects, references and symrefs (2/3)

The contents of an object depend on its type:

Blob: raw data; stores file contents.

Tree: directory contents.



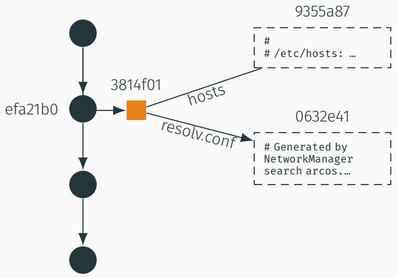
Objects, references and symrefs (2/3)

The contents of an object depend on its type:

Blob: raw data; stores file contents.

Tree: directory contents.

Commit: information about a revision.



Objects, references and symrefs (2/3)

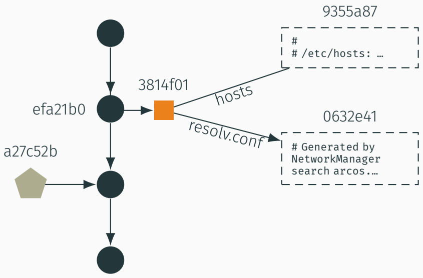
The contents of an object depend on its type:

Blob: raw data; stores file contents.

Tree: directory contents.

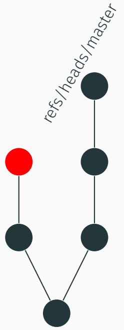
Commit: information about a revision.

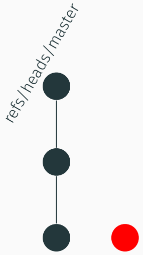
Tag: ref pointing to a commit + message + PGP signature (optional).



Objects, references and symrefs (3/3)

Typically, objects can be reached given a ref (but not always).





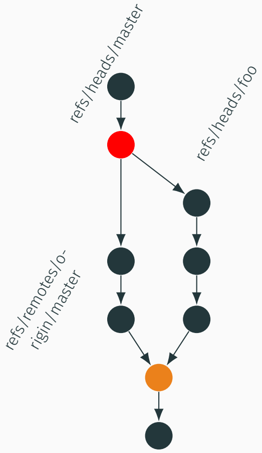
Unreachable object: an object which is not reachable from any reference.

Dangling object: not reachable even from other unrechable objects.

More at `gitglossary(7)`

Project history, branches and tags

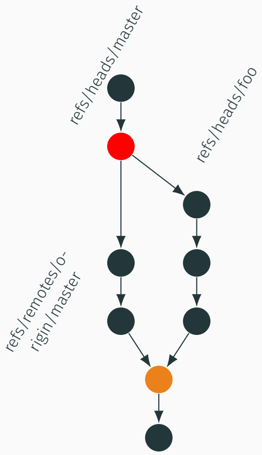
Commit objects form a DAG (they point to their parents). This DAG is known as the history of a project.



Project history, branches and tags

Commit objects form a DAG (they point to their parents). This DAG is known as the history of a project.

Branch: an active line of development; *tip:* the most recent commit.

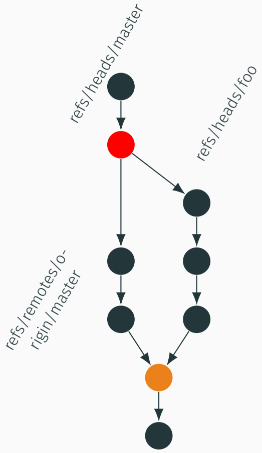


Project history, branches and tags

Commit objects form a DAG (they point to their parents). This DAG is known as the history of a project.

Branch: an active line of development; *tip:* the most recent commit.

(Branch) head: a reference to the tip of a branch.
Local heads at: `refs/heads/`.



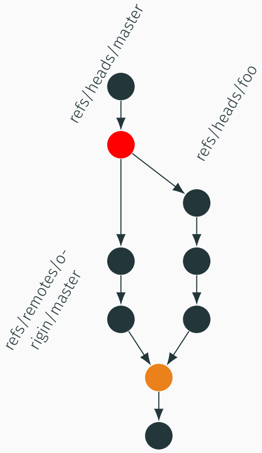
Project history, branches and tags

Commit objects form a DAG (they point to their parents). This DAG is known as the history of a project.

Branch: an active line of development; *tip:* the most recent commit.

(Branch) head: a reference to the tip of a branch.
Local heads at: `refs/heads/`.

Remote-tracking branch: a ref to a remote head; follow changes from another repository.
At `refs/remotes/*/`.

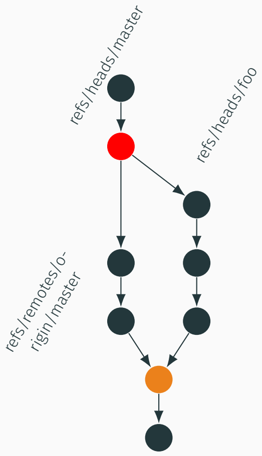


Project history, branches and tags

Commit objects form a DAG (they point to their parents). This DAG is known as the history of a project.

Merge commit: a commit object that has ≥ 2 parents.

Octopus: a merge that has > 2 parents.



The “index” (cache) file

Short story: basically, it is the staging area for the next commit.

- “A collection of files with stat information, whose contents are stored as objects.” [`gitglossary(7)`]

[1]Last modified time, size, etc.

The “index” (cache) file

Short story: basically, it is the staging area for the next commit.

- “A collection of files with stat information, whose contents are stored as objects.” [`gitglossary(7)`]
- For each file, it stores <object SHA-1> <attributes[1]>

```
100644 01cb7066623241a0e5714a6630f0355eb0c80de4 0   .gitignore
…
100644 94fbec4cf383e9122c22d60cfad91b3c897e2c63 0   slides.tex
```

[1]Last modified time, size, etc.

The “index” (cache) file

Short story: basically, it is the staging area for the next commit.

- “A collection of files with stat information, whose contents are stored as objects.” [`gitglossary(7)`]
- For each file, it stores <object SHA-1> <attributes[1]>

```
100644 01cb7066623241a0e5714a6630f0355eb0c80de4 0   .gitignore
…
100644 94fbec4cf383e9122c22d60cfad91b3c897e2c63 0   slides.tex
```

- Changes to the working tree found by comparing these attributes.

[1]Last modified time, size, etc.

The “index” (cache) file

Short story: basically, it is the staging area for the next commit.

- “A collection of files with stat information, whose contents are stored as objects.” [`gitglossary(7)`]
- For each file, it stores <object SHA-1> <attributes[1]>

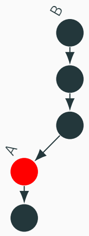
```
100644 01cb7066623241a0e5714a6630f0355eb0c80de4 0   .gitignore
…
100644 94fbec4cf383e9122c22d60cfad91b3c897e2c63 0   slides.tex
```

- Changes to the working tree found by comparing these attributes.
- Entries may be updated (`git add`) and new commits may be created from the index.

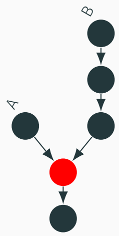
[1]Last modified time, size, etc.

Other definitions (1/3)

Fast-forward: a special type of merge; given two heads A and B, merging B into A is considered fast-forward if $merge\_base(A, B) == A$, i.e. A is ancestor of B.



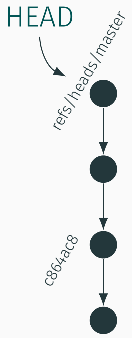
Fast-forward (update ref only!)



Non fast-forward (requires a merge)

Other definitions (2/3)

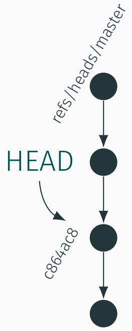
HEAD: symref that dereferences to the current checked-out head.



Other definitions (2/3)

HEAD: symref that dereferences to the current checked-out head.

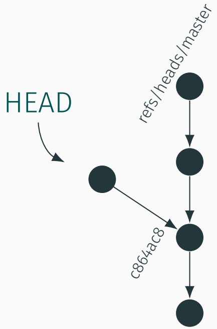
Detached HEAD: HEAD may also point at an arbitrary commit, i.e. “detached” from any branch. You may make commits in this state, but...



Other definitions (2/3)

HEAD: symref that dereferences to the current checked-out head.

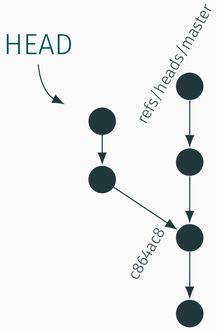
Detached HEAD: HEAD may also point at an arbitrary commit, i.e. “detached” from any branch. You may make commits in this state, but...



Other definitions (2/3)

HEAD: symref that dereferences to the current checked-out head.

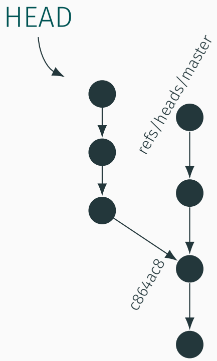
Detached HEAD: HEAD may also point at an arbitrary commit, i.e. “detached” from any branch. You may make commits in this state, but...



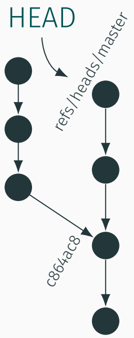
Other definitions (2/3)

HEAD: symref that dereferences to the current checked-out head.

Detached HEAD: HEAD may also point at an arbitrary commit, i.e. “detached” from any branch. You may make commits in this state, but...



Other definitions (2/3)



if HEAD is made to point somewhere else, they will become unreachable (and eventually deleted by the GC). Create a ref to avoid this!

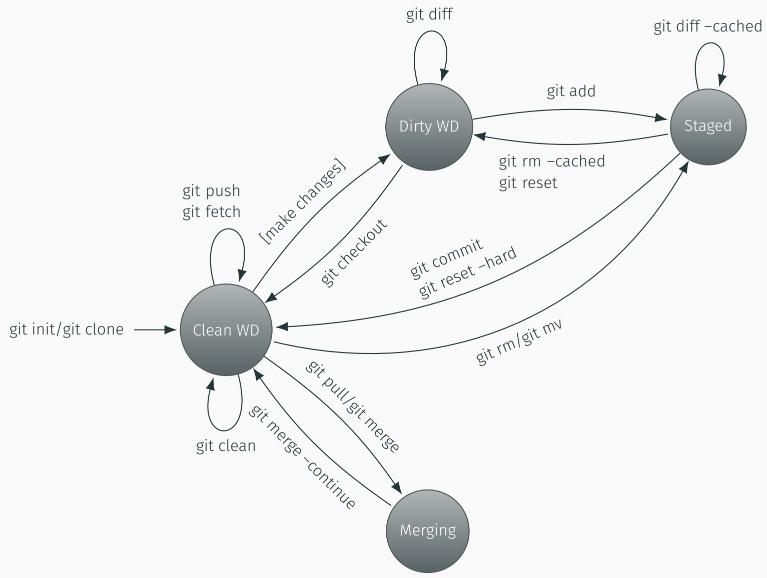
Other definitions (3/3)

Reflog: stores the local history of a ref.

- What was `HEAD` pointing at before the last change?
- What did `refs/heads/foo` pointed at two weeks ago?

[(l)user] Porcelain

Simple (and incomplete) FSM



Init/Clone a repository

To get started, you can either

- Create an empty repository, e.g.

```
$ git init [--bare] ~/foo/
```

- Obtain a copy of a remote repository[2], e.g.

```
$ git clone [--depth=1] https://earth/public/repo.git/
```

[2]The `--depth` option creates a shallow clone (history pruned). To unshallow run `git pull --unshallow`.

Overview of branches

```
# Create a branch started off from 'HEAD'
$ git branch foo HEAD
$ git checkout foo
Switched to branch 'foo'

$ git checkout -b foo HEAD # Shorthand for the above commands
```

Overview of branches

```
# Create a branch started off from 'HEAD'
$ git branch foo HEAD
$ git checkout foo
Switched to branch 'foo'

$ git checkout -b foo HEAD # Shorthand for the above commands

# Create an orphan branch (new totally disconnected history)
$ git checkout --orphan foo HEAD
```

Overview of branches

```
# Create a branch started off from 'HEAD'
$ git branch foo HEAD
$ git checkout foo
Switched to branch 'foo'

$ git checkout -b foo HEAD # Shorthand for the above commands

# Create an orphan branch (new totally disconnected history)
$ git checkout --orphan foo HEAD

$ git branch -d foo      # Delete a branch
$ git branch -m foo bar  # Move/rename a branch

# List branches
$ git branch --verbose
* foo    d7832a7f Closes issue #17
  master 99446829 Closes issue #16
```

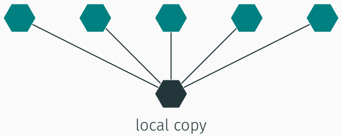
Overview of branches

```
# Create a branch started off from 'HEAD'
$ git branch foo HEAD
$ git checkout foo
Switched to branch 'foo'

$ git checkout -b foo HEAD # Shorthand for the above commands

# Create an orphan branch (new totally disconnected history)
$ git checkout --orphan foo HEAD

$ git branch -d foo      # Delete a branch
$ git branch -m foo bar  # Move/rename a branch

# List branches
$ git branch --verbose
* foo    d7832a7f Closes issue #17
  master 99446829 Closes issue #16

# Merge a branch
$ git merge foo
# Resolve conflicts + 'git add <pathspec> ' + 'git merge --continue'
# or 'git merge --abort'
```

Working with remotes (1/4)

Git can manage remote sites (remotes[3]) whose branches you track.

- Supports http[s]://, ssh://, git:// and file://.
- `git-clone` automatically adds the remote *origin* (the URL you cloned)
- May have different push/fetch URLs

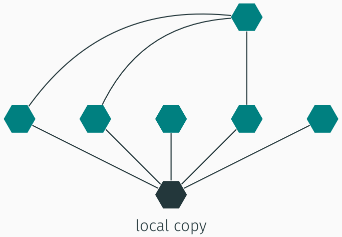


[3]See `git-remote(1)` for more information.

Working with remotes (1/4)

Git can manage remote sites (remotes[3]) whose branches you track.

- Supports http[s]://, ssh://, git:// and file://.
- `git-clone` automatically adds the remote *origin* (the URL you cloned)
- May have different push/fetch URLs
- REMEMBER: Git is distributed



[3]See `git-remote(1)` for more information.

Working with remotes (2/4)

- Remotes may be added with `git-remote`, e.g.

```
$ git remote add earth https://earth/public/repo.git/
```

  Default is to track all branches[4].

[4] Otherwise, see `-t <branch>`

Working with remotes (2/4)

- Remotes may be added with `git-remote`, e.g.

```
$ git remote add earth https://earth/public/repo.git/
```

  Default is to track all branches[4].

- `git-push` pushes refs (+ objects) to a remote, e.g.

```
$ git push earth master
```

[4]Otherwise, see `-t <branch>`

Working with remotes (2/4)

- Remotes may be added with `git-remote`, e.g.

```
$ git remote add earth https://earth/public/repo.git/
```

  Default is to track all branches[4].

- `git-push` pushes refs (+ objects) to a remote, e.g.

```
$ git push earth master
```

- `git-fetch` fetches refs (+ objects) from a remote, e.g.

```
$ git fetch earth master
```

  Fetched refs will be in `refs/remotes/earth/*`.

[4]Otherwise, see `-t <branch>`

Working with remotes (2/4)

- Remotes may be added with `git-remote`, e.g.

```
$ git remote add earth https://earth/public/repo.git/
```

  Default is to track all branches[4].

- `git-push` pushes refs (+ objects) to a remote, e.g.

```
$ git push earth master
```

- `git-fetch` fetches refs (+ objects) from a remote, e.g.

```
$ git fetch earth master
```

  Fetched refs will be in `refs/remotes/earth/*`.

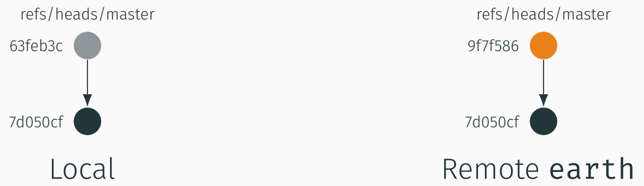
- `git-pull` is equivalent to `git fetch` + `git merge FETCH_HEAD`

[4]Otherwise, see `-t <branch>`

Working with remotes (3/4)

Trying a `$ git push earth master`

- On the remote end: receive object pack + update refs

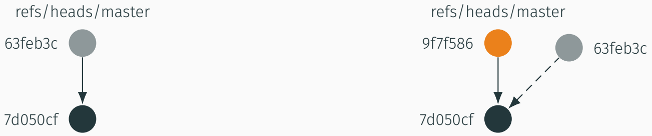


[5] See the `-f` git-push(1) option and `receive.denyNonFastForwards`.

Working with remotes (3/4)

Trying a `$ git push earth master`

- On the remote end: receive object pack + update refs
- Synching only requires commit 63feb3c



Local

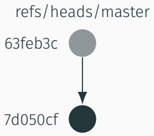
Remote **`earth`**

[5]See the `-f` git-push(1) option and `receive.denyNonFastForwards`.

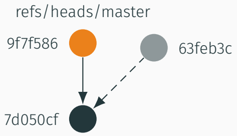
Working with remotes (3/4)

Trying a `$ git push earth master`

- On the remote end: receive object pack + update refs
- Synching only requires commit 63feb3c
- Non-FF. If earth updates refs/heads/master, commit 9f7f586 is lost!
  Typically, remotes will deny non-fast-forward pushes[5]



Local



Remote `earth`

[5] See the `-f` git-push(1) option and `receive.denyNonFastForwards`.

Working with remotes (3/4)

Trying a `$ git push earth master`

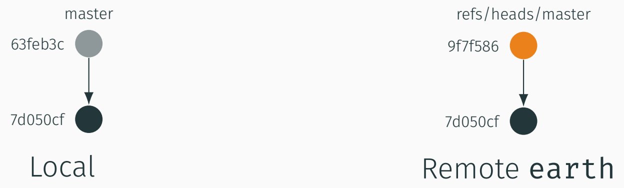
- On the remote end: receive object pack + update refs
- Synching only requires commit 63feb3c
- Non-FF. If earth updates refs/heads/master, commit 9f7f586 is lost!
  Typically, remotes will deny non-fast-forward pushes[5]

```
! [rejected]      master ->master (non-fast-forward)
error: failed to push some refs to '…'
```

[5]See the `-f` git-push(1) option and `receive.denyNonFastForwards`.

Working with remotes (4/4)

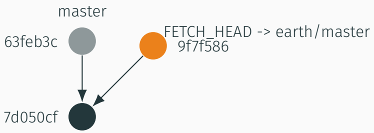
Trying a `$ git pull earth master`[6]



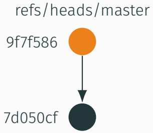
[6] The merge might be avoided; see the `--rebase` option.

Working with remotes (4/4)

Trying a `$ git pull earth master`[6]



Local

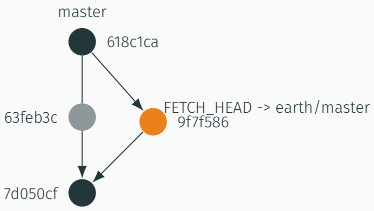


Remote `earth`

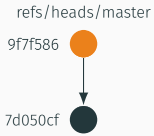
[6]The merge might be avoided; see the `--rebase` option.

Working with remotes (4/4)

Trying a `$ git pull earth master`[6]



Local



Remote `earth`

[6]The merge might be avoided; see the `--rebase` option.

Bug hunting

Git helps you to find bugs (and their authors)…

git-bisect(1) uses binary search to find a “bad” commit

```
$ git bisect start HEAD v1.2 # HEAD is bad, v1.2 is good
$ git bisect [good|bad] # Manually mark it as working/broken
…
$ git bisect run my_script arguments # Or automatically (good if $? = 0)
$ git bisect reset
```

git-blame(1) annotates each line of a file with revision information

```
$ git blame README.md
63feb3c8 (jalopezg 2019-01-18 19:36:40 +0100 1) >This file was created by …
ded8aa43 (jalopezg 2019-01-22 20:18:04 +0100 2) foo
```

Specifying revisions (1/2)

Some Git commands take symbolic revision parameters (names specific commit or all commits reachable from that commit)[7].

<sha1> SHA-1 object name, or a non-ambiguous leading substring.

<refname> A ref name, e.g. refs/heads/master. Search order: `$GIT_DIR/<refname>`, `refs/`, `refs/tags/`, `refs/heads/`, `refs/remotes/`, `refs/remotes/<refname>/HEAD`.

<refname>@{<n>} The n-th prior value of that ref.

<rev>^ The first parent.

<rev>~<n> The n-th generation ancestor.

<rev>:<path> Names the blob or tree of <rev>.

[7] This is an overview; see gitrevisions(7) for the complete list.

Specifying revisions (2/2)

Specifying ranges:

^<rev> Exclude commits reachable from <rev>.

<rev1>..<rev2> A shorthand for ^rev1 rev2, i.e. commits reachable from rev2, but not from rev1, or $(rev1, \ rev2]$

<rev1>...<rev2> Commits reachable either from rev1 or rev2, but not from both.

[sudoer] More porcelain

Save/Restore a dirty working directory

`git-stash(1)` saves the current state of the working directory + the index, and goes back to a clean WD.

Saved changes can be restored with `$ git stash pop`. Git-stash stack can be dumped by `$ git stash list`.

Save/Restore a dirty working directory

`git-stash(1)` saves the current state of the working directory + the index, and goes back to a clean WD.

Saved changes can be restored with `$ git stash pop`. Git-stash stack can be dumped by `$ git stash list`.

```
$ echo foo > README.md
$ git status
On branch foo
Changes not staged for commit:
        modified: README.md
```

Save/Restore a dirty working directory

`git-stash(1)` saves the current state of the working directory + the index, and goes back to a clean WD.

Saved changes can be restored with `$ git stash pop`. Git-stash stack can be dumped by `$ git stash list`.

```
$ echo foo > README.md
$ git status
On branch foo
Changes not staged for commit:
        modified: README.md

$ git stash
Saved working directory and index state WIP on foo: 9f7f586 README.md has been added
$ git status
On branch foo
nothing to commit, working tree clean
```

Save/Restore a dirty working directory

`git-stash(1)` saves the current state of the working directory + the index, and goes back to a clean WD.

Saved changes can be restored with `$ git stash pop`. Git-stash stack can be dumped by `$ git stash list`.

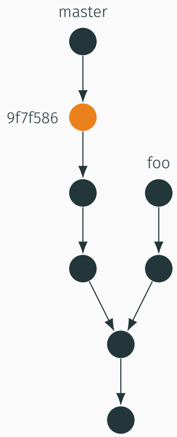
```
$ echo foo > README.md
$ git status
On branch foo
Changes not staged for commit:
        modified: README.md

$ git stash
Saved working directory and index state WIP on foo: 9f7f586 README.md has been added
$ git status
On branch foo
nothing to commit, working tree clean

$ git stash pop
On branch foo
Changes not staged for commit:
        modified: README.md
Dropped refs/stash@{0} (35365e0c188e877ded1ecdd8190ec5bb1b6c2c1b)
```

Applying changes from other branches

`git-cherry-pick(1)` apply the changes introduced by the given commits, e.g.

`$ git cherry-pick 9f7f586`.

The patch may not apply cleanly; if that is the case, you are required to resolve conflicts

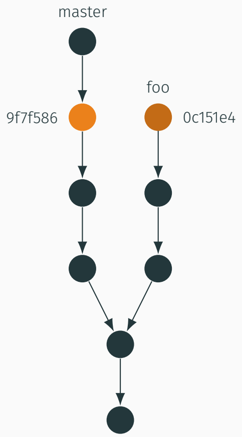


Applying changes from other branches

`git-cherry-pick(1)` apply the changes introduced by the given commits, e.g.

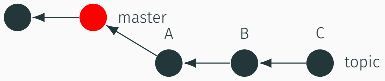
`$ git cherry-pick 9f7f586`.

The patch may not apply cleanly; if that is the case, you are required to resolve conflicts



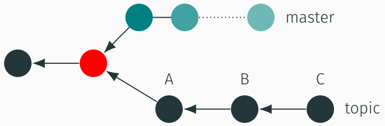
Rebase (+ interactive rebase!) (1/4)

Sometimes you fork a branch and it becomes outdated w.r.t. its parent. Quite probably, you would merge the parent branch.



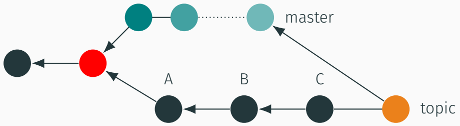
Rebase (+ interactive rebase!) (1/4)

Sometimes you fork a branch and it becomes outdated w.r.t. its parent. Quite probably, you would merge the parent branch.



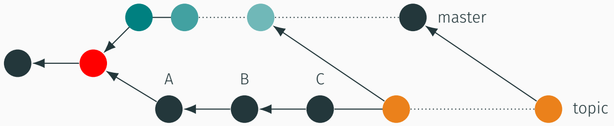
Rebase (+ interactive rebase!) (1/4)

Sometimes you fork a branch and it becomes outdated w.r.t. its parent. Quite probably, you would merge the parent branch.



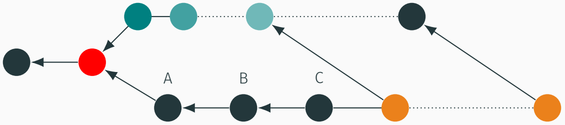
Rebase (+ interactive rebase!) (1/4)

Sometimes you fork a branch and it becomes outdated w.r.t. its parent. Quite probably, you would merge the parent branch.

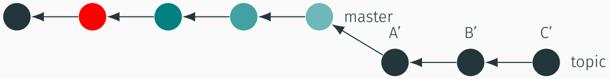


Rebase (+ interactive rebase!) (1/4)

Sometimes you fork a branch and it becomes outdated w.r.t. its parent. Quite probably, you would merge the parent branch.



This clutters project history. Reapplying `topic` commits on top of `master` is better!



```
# Assuming that 'topic' is the current branch, this gives the result above
$ git rebase master
```

Rebase (+ interactive rebase!) (2/4)

It is one of the most powerful Git commands. In fact, it can be used to rewrite project history (next slide).

Rebase (+ interactive rebase!) (2/4)

It is one of the most powerful Git commands. In fact, it can be used to rewrite project history (next slide).

If there are conflicts, you will have to resolve them (as in merge).

Rebase (+ interactive rebase!) (2/4)

It is one of the most powerful Git commands. In fact, it can be used to rewrite project history (next slide).

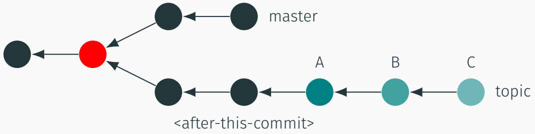
If there are conflicts, you will have to resolve them (as in merge).

⚠ `GIT-REBASE(1)` IMPLICATIONS:

- Requires rewriting commits and is PROBLEMATIC if you already pushed those objects
- You can break things: YOU HAVE BEEN WARNED!
- If you ever force-push a rebased branch, others will have to fix their history. See git-rebase(1), section "RECOVERING FROM UPSTREAM REBASE".

Rebase (+ interactive rebase!) (3/4)

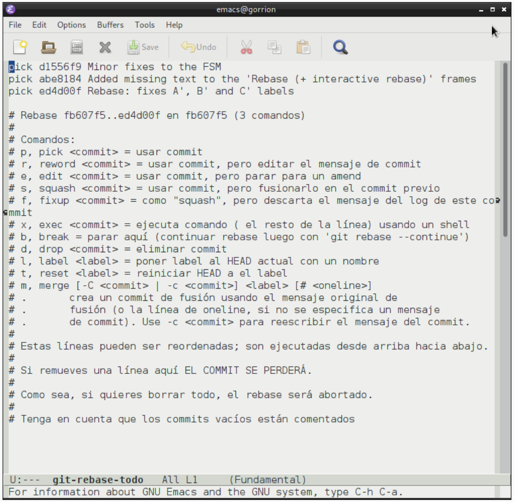
`git-rebase(1)` has an interactive mode in which you can edit/reorder/remove the commits which are rebased.
It is very common to rewrite part of a branch to have a more meaningful history, e.g.



```
# This fires up an editor and gives you the chance to edit the commit list before
     they are applied (commits A, B and C)
$ git rebase -i <after-this-commit>
```

⚠ USE WITH CARE. Read git-rebase implications!

Rebase (+ interactive rebase!) (4/4)



```
pick d1556f9 Minor fixes to the FSM
pick abe8184 Added missing text to the 'Rebase (+ interactive rebase)' frames
pick ed4d00f Rebase: fixes A', B' and C' labels

# Rebase fb607f5..ed4d00f en fb607f5 (3 comandos)
#
# Comandos:
# p, pick <commit> = usar commit
# r, reword <commit> = usar commit, pero editar el mensaje de commit
# e, edit <commit> = usar commit, pero parar para un amend
# s, squash <commit> = usar commit, pero fusionarlo en el commit previo
# f, fixup <commit> = como "squash", pero descarta el mensaje del log de este co
mmit
# x, exec <commit> = ejecuta comando ( el resto de la línea) usando un shell
# b, break = parar aquí (continuar rebase luego con 'git rebase --continue')
# d, drop <commit> = eliminar commit
# l, label <label> = poner label al HEAD actual con un nombre
# t, reset <label> = reiniciar HEAD a el label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       crea un commit de fusión usando el mensaje original de
# .       fusión (o la línea de oneline, si no se especifica un mensaje
# .       de commit). Use -c <commit> para reescribir el mensaje del commit.
#
# Estas líneas pueden ser reordenadas; son ejecutadas desde arriba hacia abajo.
#
# Si remueves una línea aquí EL COMMIT SE PERDERÁ.
#
# Como sea, si quieres borrar todo, el rebase será abortado.
#
# Tenga en cuenta que los commits vacíos están comentados

U:---  git-rebase-todo   All L1     (Fundamental)
For information about GNU Emacs and the GNU system, type C-h C-a.
```

More about fixing history (1/2)

So common that `git-commit(1)` has the `--squash` and `--fixup` options. They mark commits to be automatically squashed. Rewriting occurs after a `$ git rebase --autosquash`.

More about fixing history (1/2)

So common that `git-commit(1)` has the `--squash` and `--fixup` options. They mark commits to be automatically squashed. Rewriting occurs after a `$ git rebase --autosquash`.

```
$ git log --oneline
e7a2019 (HEAD -> master) Any other changes
9f7f586 Added README.md
02a7fb9 Added bar.txt
```

More about fixing history (1/2)

So common that `git-commit(1)` has the `--squash` and `--fixup` options. They mark commits to be automatically squashed. Rewriting occurs after a `$ git rebase --autosquash`.

```
$ git log --oneline
e7a2019 (HEAD -> master) Any other changes
9f7f586 Added README.md
02a7fb9 Added bar.txt

$ echo foo >> README.md && git commit -a --fixup 9f7f586
$ git log --oneline
24a54df (HEAD -> master) fixup! Added README.md
e7a2019 Any other changes
9f7f586 Added README.md
02a7fb9 Added bar.txt
```

More about fixing history (1/2)

So common that `git-commit(1)` has the `--squash` and `--fixup` options. They mark commits to be automatically squashed. Rewriting occurs after a `$ git rebase --autosquash`.

```
$ git log --oneline
e7a2019 (HEAD -> master) Any other changes
9f7f586 Added README.md
02a7fb9 Added bar.txt

$ echo foo >> README.md && git commit -a --fixup 9f7f586
$ git log --oneline
24a54df (HEAD -> master) fixup! Added README.md
e7a2019 Any other changes
9f7f586 Added README.md
02a7fb9 Added bar.txt

$ git rebase -i --autosquash 02a7fb9
Successfully rebased and updated refs/heads/master.
$ git log --oneline
528efb7 (HEAD -> master) Any other changes
a59735c Added README.md
02a7fb9 Added bar.txt
```

⚠ USE WITH CARE. Read git-rebase implications!

More about fixing history (2/2)

If you only need to rewrite the last commit use

```
$ git commit --amend
```

⚠ USE WITH CARE. Read git-rebase implications!

git filter-branch

Q: I know how to rewrite commits. Can I automate the process?

git filter-branch

Q: I know how to rewrite commits. Can I automate the process?
A: `git-filter-branch(1)` lets you rewrite branches, applying filters to modify each tree/information about each commit, e.g.

```
$ git log --oneline
92cb761 (HEAD -> foo) Added nsswitch.conf
9f7f586 Added README.md
02a7fb9 (bar) Added bar.txt

$ git filter-branch --msg-filter 'sed -e "s/Added \([[:graph:]]*\)$/\1 has
been added/"' foo

$ git log --oneline
6e9fbd6 (HEAD -> foo) nsswitch.conf has been added
63feb3c README.md has been added
2fe54f3 bar.txt has been addded
```

⚠ USE WITH CARE. Read git-rebase implications!

Comparing branches

Q: Can I see where each of the given branches is w.r.t. others?

Comparing branches

Q: Can I see where each of the given branches is w.r.t. others?

A: `git-show-branch` is your friend. Also, `git log --graph --oneline …`

```
$ git show-branch master foo
! [master] Added README.md
 * [foo] Added nsswitch.conf
  ! [bar] Added bar.txt
---
 *  [foo] Added nsswitch.conf
+*  [master] Added README.md
+*+ [bar] Added bar.txt

# To include all remote-tracking and local branches:
$ git show-branch --all
```

Share by other means

TAR or ZIP archives of a particular tree can be created by `git-archive(1)`, e.g.

```
$ git archive --format=tar --prefix=foo/ -o foo.tar.gz master
```

Git also can generate an archive of packed objects and references to be imported into a repository (useful if machines are not directly connected), e.g.

```
[alice@earth ~]$ git bundle create /tmp/foo-master.git master
# /tmp/foo-master.git is copied to moon by some means.

[bob@moon ~]$ git clone -b master ~/foo-master.git

# Or if the repository already exists…
[bob@moon ~]$ git remote add foo-bundle ~/foo-master.git
[bob@moon ~]$ git pull foo-bundle master
```

ReReRe

"git-rerere - Reuse recorded resolution of conflicted merges"

FYI, see the `git-rerere(1)` manual page.

[r00t] Plumbing

Repository layout

objects/: the object store.

objects/[0-9a-f][0-9a-f]/: loose objects.

objects/pack/: object packs (store many objects in compressed form).

refs/: references are stored in subdirectories of this directory.

packed-refs: the same as refs/ but in a more efficient way.

HEAD: the HEAD symref.

```
repository/
└── .git/
    ├── objects/
    │   ├── pack/
    │   └── [0-9a-f][0-9a-f]/
    ├── refs/
    │   ├── heads/
    │   ├── tags/
    │   └── remotes/
    ├── packed-refs
    ├── HEAD
    ├── config
    ├── hooks/
    ├── index
    ├── info/
    ├── logs/
    ├── shallow
    └── …
└── …
```

More at `gitrepository-layout(5)`.

Repository layout

config: repository specific configuration file.

hooks/: (described later).

index: the “index” file.

info/: additional information, e.g. `info/grafts`.

logs/: reflogs are stored here.

shallow: similar to `info/grafts` but internally used for shallow clones.

```
repository/
├── .git/
│   ├── objects/
│   │   ├── pack/
│   │   └── [0-9a-f][0-9a-f]/
│   ├── refs/
│   │   ├── heads/
│   │   ├── tags/
│   │   └── remotes/
│   ├── packed-refs
│   ├── HEAD
│   ├── config
│   ├── hooks/
│   ├── index
│   ├── info/
│   ├── logs/
│   ├── shallow
│   └── …
└── …
```

More at `gitrepository-layout(5)`.

HOWTO: create a commit using the “index”

```
$ echo foo > bar.txt

# Add 'bar.txt' to the index
$ git update-index --add bar.txt
```

HOWTO: create a commit using the “index”

```
$ echo foo > bar.txt

# Add 'bar.txt' to the index
$ git update-index --add bar.txt

# Create a tree object from the current index
$ git write-tree
6d21ed3d662ea6040da2fe0fd66fe80fefe689a5
```

HOWTO: create a commit using the “index”

```
$ echo foo > bar.txt

# Add 'bar.txt' to the index
$ git update-index --add bar.txt

# Create a tree object from the current index
$ git write-tree
6d21ed3d662ea6040da2fe0fd66fe80fefe689a5

# Create a new commit object
$ git commit-tree -p HEAD -m 'Added␣bar.txt' 6d21ed3d662ea6040da2fe0fd66fe80fefe689a5
02a7fb9f9145086807cbe2ed45ea82149c3d1b34
```

HOWTO: create a commit using the “index”

```
$ echo foo > bar.txt

# Add 'bar.txt' to the index
$ git update-index --add bar.txt

# Create a tree object from the current index
$ git write-tree
6d21ed3d662ea6040da2fe0fd66fe80fefe689a5

# Create a new commit object
$ git commit-tree -p HEAD -m 'Added␣bar.txt' 6d21ed3d662ea6040da2fe0fd66fe80fefe689a5
02a7fb9f9145086807cbe2ed45ea82149c3d1b34

# Update refs/heads/master
$ git update-ref refs/heads/master 02a7fb9f9145086807cbe2ed45ea82149c3d1b34
```

HOWTO: create a commit using the “index”

```
$ echo foo > bar.txt

# Add 'bar.txt' to the index
$ git update-index --add bar.txt

# Create a tree object from the current index
$ git write-tree
6d21ed3d662ea6040da2fe0fd66fe80fefe689a5

# Create a new commit object
$ git commit-tree -p HEAD -m 'Added␣bar.txt' 6d21ed3d662ea6040da2fe0fd66fe80fefe689a5
02a7fb9f9145086807cbe2ed45ea82149c3d1b34

# Update refs/heads/master
$ git update-ref refs/heads/master 02a7fb9f9145086807cbe2ed45ea82149c3d1b34

$ git log -1
commit 02a7fb9f9145086807cbe2ed45ea82149c3d1b34 (HEAD -> master)
Author: Javier López Gómez <jalopezg@inf.uc3m.es>
Date:   Fri Jan 18 18:59:39 2019 +0100

    Added bar.txt
```

HOWTO: commit arbitrary content

```
# Create blob object for 'README.md'; use 'git cat-file blob 1f6c266' to see blob
     contents
$ git hash-object -t blob -w --path=README.md --stdin <<EOF
> This file was created by git-hash-object.
EOF
1f6c2663d33465dcd83f2151b15fb57369f29570
```

HOWTO: commit arbitrary content

```
# Create blob object for 'README.md'; use 'git cat-file blob 1f6c266' to see blob
     contents
$ git hash-object -t blob -w --path=README.md --stdin <<EOF
> This file was created by git-hash-object.
EOF
1f6c2663d33465dcd83f2151b15fb57369f29570

# Create tree object (add 'README.md' entry to the HEAD tree)
$ git ls-tree HEAD | awk '{␣print;␣}␣END␣{␣print␣"100644␣blob␣1
     f6c2663d33465dcd83f2151b15fb57369f29570\tREADME.md";␣}' | git mktree
0082679644a2b435b6cf09a65324292da28a41b4
```

HOWTO: commit arbitrary content

```
# Create blob object for 'README.md'; use 'git cat-file blob 1f6c266' to see blob
     contents
$ git hash-object -t blob -w --path=README.md --stdin <<EOF
> This file was created by git-hash-object.
EOF
1f6c2663d33465dcd83f2151b15fb57369f29570

# Create tree object (add 'README.md' entry to the HEAD tree)
$ git ls-tree HEAD | awk '{␣print;␣}␣END␣{␣print␣"100644␣blob␣1
     f6c2663d33465dcd83f2151b15fb57369f29570\tREADME.md";␣}' | git mktree
0082679644a2b435b6cf09a65324292da28a41b4

# Create a new commit object
$ git commit-tree -p HEAD -m 'Added␣README.md' 0082679644
     a2b435b6cf09a65324292da28a41b4
9f7f586f952c515893dd6597936f6fea64dd17ce
```

HOWTO: commit arbitrary content

```
# Create blob object for 'README.md'; use 'git cat-file blob 1f6c266' to see blob
     contents
$ git hash-object -t blob -w --path=README.md --stdin <<EOF
> This file was created by git-hash-object.
EOF
1f6c2663d33465dcd83f2151b15fb57369f29570

# Create tree object (add 'README.md' entry to the HEAD tree)
$ git ls-tree HEAD | awk '{␣print;␣}␣END␣{␣print␣"100644␣blob␣1
     f6c2663d33465dcd83f2151b15fb57369f29570\tREADME.md";␣}' | git mktree
0082679644a2b435b6cf09a65324292da28a41b4

# Create a new commit object
$ git commit-tree -p HEAD -m 'Added␣README.md' 0082679644
     a2b435b6cf09a65324292da28a41b4
9f7f586f952c515893dd6597936f6fea64dd17ce

# Update refs/heads/master
$ git update-ref refs/heads/master 9f7f586f952c515893dd6597936f6fea64dd17ce
```

HOWTO: commit arbitrary content

```
# Create blob object for 'README.md'; use 'git cat-file blob 1f6c266' to see blob
     contents
$ git hash-object -t blob -w --path=README.md --stdin <<EOF
> This file was created by git-hash-object.
EOF
1f6c2663d33465dcd83f2151b15fb57369f29570

# Create tree object (add 'README.md' entry to the HEAD tree)
$ git ls-tree HEAD | awk '{␣print;␣}␣END␣{␣print␣"100644␣blob␣1
     f6c2663d33465dcd83f2151b15fb57369f29570\tREADME.md";␣}' | git mktree
0082679644a2b435b6cf09a65324292da28a41b4

# Create a new commit object
$ git commit-tree -p HEAD -m 'Added␣README.md' 0082679644
     a2b435b6cf09a65324292da28a41b4
9f7f586f952c515893dd6597936f6fea64dd17ce

# Update refs/heads/master
$ git update-ref refs/heads/master 9f7f586f952c515893dd6597936f6fea64dd17ce

# WTF?
$ git status
On branch master
Changes to be committed:
  (use "git␣reset␣HEAD␣<file>..." to unstage)

        deleted:    README.md
```

HOWTO: commit arbitrary content

```
# Create blob object for 'README.md'; use 'git cat-file blob 1f6c266' to see blob
     contents
$ git hash-object -t blob -w --path=README.md --stdin <<EOF
> This file was created by git-hash-object.
EOF
1f6c2663d33465dcd83f2151b15fb57369f29570

# Create tree object (add 'README.md' entry to the HEAD tree)
$ git ls-tree HEAD | awk '{␣print;␣}␣END␣{␣print␣"100644␣blob␣1
     f6c2663d33465dcd83f2151b15fb57369f29570\tREADME.md";␣}' | git mktree
0082679644a2b435b6cf09a65324292da28a41b4

# Create a new commit object
$ git commit-tree -p HEAD -m 'Added␣README.md' 0082679644
     a2b435b6cf09a65324292da28a41b4
9f7f586f952c515893dd6597936f6fea64dd17ce

# Update refs/heads/master
$ git update-ref refs/heads/master 9f7f586f952c515893dd6597936f6fea64dd17ce

# WTF?
$ git status
On branch master
Changes to be committed:
  (use "git␣reset␣HEAD␣<file>..." to unstage)

        deleted:    README.md
$ git reset --hard HEAD
```

Additional stuff

Configuration (1/2)

- 480+ options. Git searches configuration at:
  - **/etc/gitconfig** System-wide configuration.
  - **~/.gitconfig** User-specific configuration.
  - **$GIT_DIR/config** Repository specific.
- Can be edited manually or using `git-config(1)`, e.g.

```
$ git config [--system|--global|--local] user.email 'John Doe'
```

```
[user]
email = jalopezg@inf.uc3m.es
name = Javier López-Gómez
…
```

Configuration (2/2)

`alias.*` options may be used to create command aliases, e.g.

```
$ git config --global alias.sb 'show-branch @ @{push}'
$ git sb
! [@] Updated README.md
 ! [@{push}] Closes issue #16
--
+  [@] …
```

FYI, see the `git-config(1)` manual page.

Fsck and garbage collection

git-fsck(1) Verifies the connectivity and validity of the objects.

```
$ git fsck [--unreachable] [--no-reflogs] [--lost-found] […]
```

git-gc(1) Runs housekeeping tasks, e.g. pack objects/refs, remove unreachable objects, prune reflog, etc.[8]

```
$ git gc [--aggressive] [--auto] […]
```

[8] `git gc --auto` may automatically run as part of some git commands.

Hooks (1/2)

Hooks are programs that are executed at certain points, e.g. after a merge *(post-merge)*, or before git-receive-pack updates refs *(pre-receive)*.

- Invoked locally/on the remote end[9]

[9] Stdout and stderr are forwarded.

Hooks (1/2)

Hooks are programs that are executed at certain points, e.g. after a merge *(post-merge)*, or before git-receive-pack updates refs *(pre-receive)*.

- Invoked locally/on the remote end[9]
- Must be executable (+x)

[9]Stdout and stderr are forwarded.

Hooks (1/2)

Hooks are programs that are executed at certain points, e.g. after a merge *(post-merge)*, or before git-receive-pack updates refs *(pre-receive)*.

- Invoked locally/on the remote end[9]
- Must be executable (+x)
- IN: environment, command-line arguments, stdin
  OUT: stdout, stderr, exit status

[9] Stdout and stderr are forwarded.

Hooks (1/2)

Hooks are programs that are executed at certain points, e.g. after a merge *(post-merge)*, or before git-receive-pack updates refs *(pre-receive)*.

- Invoked locally/on the remote end[9]
- Must be executable (+x)
- IN: environment, command-line arguments, stdin
  OUT: stdout, stderr, exit status
- Can be used for commit validation, issue management or triggering a build (CI)

[9] Stdout and stderr are forwarded.

Hooks (2/2)

See templates installed into `.git/hooks/` and the `githooks(5)` manual page.

```
applypatch-msg         pre-rebase             post-update
pre-applypatch         post-checkout          push-to-checkout
post-applypatch        post-merge             pre-auto-gc
pre-commit             pre-push               post-rewrite
prepare-commit-msg     pre-receive            sendemail-validate
commit-msg             update                 fsmonitor-watchman
post-commit            post-receive           p4-pre-submit
```

git-daemon, git-instaweb

"git-daemon - A really simple server for Git repositories" [`git-daemon(1)`], e.g.[10]

```
[alice@earth ~]$ git daemon --verbose --base-path=$HOME/repos \ --reuseaddr
--export-all $HOME/repos/*/.git


[bob@mars ~]$ git clone git://earth/foo
```

git-instaweb allows browsing a repository[11], e.g.

```
$ git instaweb [--local] --httpd=lighttpd --port=8080
$ git instaweb --stop
```

[10] It normally listens on port TCP 9418.
[11] Requires perl-cgi and lighttpd.

Other projects

git-annex

libgit2

git-crypt

Conclusion

Closing words

- Git is powerful. REALLY!
- Although targetted to SCM, it may be used to store (large) binary data and replicate it to remote sites
- Sysadmins: start versioning `/etc` today
- Read more at: `http://git-scm.org/` or `git-*(1)` manual pages
- "I am now a git expert... Am I?"

Wait! There is more...

Porcelain

git-add
git-am
git-archive
git-bisect
git-branch
git-bundle
git-checkout
git-cherry-pick
git-citool
git-clean
git-clone
git-commit
git-describe
git-diff
git-fetch
git-format-patch
git-gc
git-grep
git-gui
git-init
git-log
git-merge
git-mv
git-notes
git-pull
git-push
git-range-diff
git-rebase
git-reset
git-revert
git-rm
git-shortlog
git-show
git-stash
git-status
git-submodule
git-tag
git-worktree
git-config
git-fast-export
git-fast-import
git-filter-branch
git-mergetool
git-pack-refs
git-prune
git-reflog
git-remote
git-repack
git-replace
git-annotate
git-blame
git-count-objects
git-difftool
git-fsck
git-help
git-instaweb
git-merge-tree
git-rerere
git-show-branch
git-verify-commit
git-verify-tag
git-whatchanged
git-archimport
git-cvsexportcommit
git-cvsimport
git-cvsserver
git-imap-send
git-p4
git-quiltimport
git-request-pull
git-send-email
git-svn

Plumbing

git-apply
git-checkout-index
git-commit-graph
git-commit-tree
git-hash-object
git-index-pack
git-merge-file
git-merge-index
git-mktag
git-mktree
git-multi-pack-index
git-pack-objects
git-prune-packed
git-read-tree
git-symbolic-ref
git-unpack-objects
git-update-index
git-update-ref
git-write-tree
git-cat-file
git-cherry
git-diff-files
git-diff-index
git-diff-tree
git-for-each-ref
git-get-tar-commit-id
git-ls-files
git-ls-remote
git-ls-tree
git-merge-base
git-name-rev
git-pack-redundant
git-rev-list
git-rev-parse
git-show-index
git-show-ref
git-unpack-file
git-var
git-verify-pack
git-daemon
git-fetch-pack
git-http-backend
git-send-pack
git-update-server-info
git-http-fetch
git-http-push
git-parse-remote
git-receive-pack
git-shell
git-upload-archive
git-upload-pack
git-check-attr
git-check-ignore
git-check-mailmap
git-check-ref-format
git-column
git-credential
git-credential-cache
git-credential-store
git-fmt-merge-msg
git-interpret-trailers
git-mailinfo
git-mailsplit
git-merge-one-file
git-patch-id
git-sh-i18n
git-sh-setup
git-stripspace

Thanks!

Thank you
for listening!

¿?