

Git

A Distributed Version Control System

Carlos García Campos
carlosgc@gsync.es



A couple of Quotes

“For the first 10 years of kernel maintenance, we literally used tarballs and patches, which is a much superior source code management system than CVS is[...].” – Linus Torvalds

“When I say I hate CVS with a passion, I have to also say that if there any SVN users in the audience, you might want to leave. Because my hatred of CVS has meant that I see Subversion as being the most pointless project ever started, because the whole slogan for the Subversion for a while was ‘CVS done right’ or something like that. And if you start with that kind of slogan, there is nowhere you can go. It’s like, there is no way to do CVS right.” – Linus Torvalds



First of all

- Telling git who you are

```
[user]
```

```
name = Your Name
```

```
email = your@mail.ext
```

- Global: `/.gitconfig`
- Repository specific: `repodir/.git/config`
- Extensible config file
- `git-config` command



Basic operations

- **Creating/Initializing a new repository**

```
$ mkdir project
```

```
$ cd project
```

```
$ git init
```

- **Committing**

```
$ git add file
```

```
$ git commit
```

- **Differences**

- `git diff`: differences between the work tree and the index

- `git diff --cached`: differences between HEAD and the index

- **Special shortcut for committing**

```
$ git commit -a
```

- **Status**

```
$ git status
```

NOTE: the index is a snapshot of the working tree that represents the content that will be affected by a commit



Fixing mistakes

- Resetting by changing the history

```
$ git reset --hard <commit>
```

- Reverting with a new commit

```
$ git revert <commit>
```

- Getting an old version of a file

```
$ git checkout <commit> path/to/file
```



Exploring the history

- *git log*

```
commit ce6ddeac2200225b173c52e2509ffcc01b2b4a1d
Author: Carlos Garcia Campos <carlosgc@gnome.org>
Date:   Mon Nov 19 16:14:01 2007 +0100
```

Initial commit

- Commit: commit id (sha1)
- Author: commit author (not the committer)
- Date: commit date
- Commit message: (a single line briefly describing the change and, optionally, a blank line followed by one or more lines with a detailed description)

ChangeLog file is no longer needed.



Customizing the log command output

- Change stats: `-stat`
- Diff of the changes (patch): `-p`
- With information about the altered paths: `-name-status`
- Wit full information about both the Author and Committer:
`-pretty=fuller`
- Limiting the amount of commits shown : `-n<n_commits>`
- Useful shortcut: `git show`

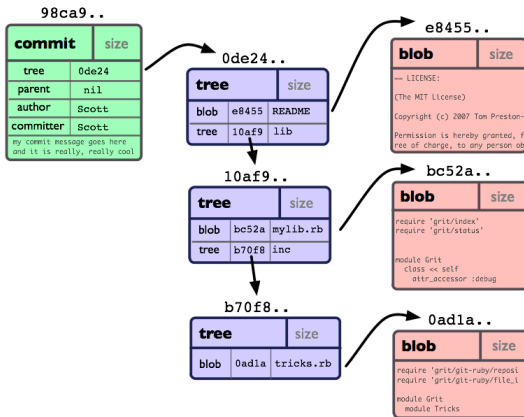


Git internals

- The whole history of the repository is stored by Git in files referenced by its contents (SHA1) rather than a file name.
- Such files are the “objects” that form the Git Object Model.
- Every object consists of a type, a size and its contents.
- Four types of objects:
 - **blob** is just a file
 - **tree** is a container of pointers to blobs and other tree objects
 - **commit** is a link to a physical state of a tree with a description of how we got there and why
 - **tag** is used to mark commit objects with a label



The Git Object Model (Example)



Working with branches

- Branches are quite “cheap” (fast and clean process)

- Creating a new branch

```
$ git branch <branch> [<start-point>]
```

- Listing available branches

```
$ git branch
```

- Changing the current branch

```
$ git checkout
```

- Creating a branch and changing to it in a single operation

```
$ git checkout -b <branch> [<start-point>]
```

- Merging branches

```
$ git merge <branch>
```

- Updating the current branch from another

```
$ git rebase <branch>
```



Pulling

- Cloning a repository (clone != checkout)

```
$ git clone <uri>
```

- Getting updates

```
$ git fetch <uri>
```

```
$ git merge origin/master
```

- Useful shortcut (commonly used)

```
$ git pull
```

- Updating a remote repository

```
$ git push
```



Workflow (Summary)

```
$ git clone <uri>
$ cd project
$ git checkout -b new-feature
(some changes, git add, etc.)
$ git commit -a
(more changes)
$ git commit -a
(last changes, my new feature is now ready)
$ git commit -a
$ git checkout master
$ git pull
$ git checkout new-feature
$ git rebase master
$ git checkout master
$ git merge new-feature
$ git push origin master
```



Big advantages for the daily work

- **Offline work**
 - Working on more than one feature at the same time
 - Micro-commits
 - Easy sharing of code
 - Author != Committer
 - Better history
 - Interactivity with other systems (git-svn, git-cvs)
 - **Performance!!!**



Big advantages for the daily work

- Offline work
- Working on more than one feature at the same time
- Micro-commits
- Easy sharing of code
- Author != Committer
- Better history
- Interactivity with other systems (git-svn, git-cvs)
- **Performance!!!**



Big advantages for the daily work

- Offline work
- Working on more than one feature at the same time
- Micro-commits
- Easy sharing of code
- Author != Committer
- Better history
- Interactivity with other systems (git-svn, git-cvs)
- **Performance!!!**



Big advantages for the daily work

- Offline work
- Working on more than one feature at the same time
- Micro-commits
- Easy sharing of code
- Author != Committer
- Better history
- Interactivity with other systems (git-svn, git-cvs)
- **Performance!!!**



Big advantages for the daily work

- Offline work
- Working on more than one feature at the same time
- Micro-commits
- Easy sharing of code
- Author != Committer
- Better history
- Interactivity with other systems (git-svn, git-cvs)
- **Performance!!!**



Big advantages for the daily work

- Offline work
- Working on more than one feature at the same time
- Micro-commits
- Easy sharing of code
- Author != Committer
- Better history
- Interactivity with other systems (git-svn, git-cvs)
- **Performance!!!**



Big advantages for the daily work

- Offline work
- Working on more than one feature at the same time
- Micro-commits
- Easy sharing of code
- Author != Committer
- Better history
- Interactivity with other systems (git-svn, git-cvs)
- Performance!!!



Big advantages for the daily work

- Offline work
- Working on more than one feature at the same time
- Micro-commits
- Easy sharing of code
- Author != Committer
- Better history
- Interactivity with other systems (git-svn, git-cvs)
- **Performance!!!**



Some tips and tricks

- Cherry-picking

```
$ git cherry-pick <commit>
```

- Move this away for a while, I need to fix something first

```
$ git stash "work in progress for foo thing"  
changes, commit, push and whatever  
$ git stash apply
```

- Merge this, but please, do not change the history because I have some embarrassing private commits

```
$ git merge --squash <branch>  
$ git commit -a  
$ git push
```

- Looking for a regression

```
$ git bisect start  
$ git bisect good <commit-id>  
$ git bisect bad <commit-id>  
$ git bisect [good|bad]  
(repeat until you find the guilty commit  
telling git if current version is good or bad)  
$ git bisect reset
```



Some tips and tricks

- Cherry-picking

```
$ git cherry-pick <commit>
```

- Move this away for a while, I need to fix something first

```
$ git stash "work in progress for foo thing"  
changes, commit, push and whatever  
$ git stash apply
```

- Merge this, but please, do not change the history because I have some embarrassing private commits

```
$ git merge --squash <branch>  
$ git commit -a  
$ git push
```

- Looking for a regression

```
$ git bisect start  
$ git bisect good <commit-id>  
$ git bisect bad <commit-id>  
$ git bisect [good|bad]  
(repeat until you find the guilty commit  
telling git if current version is good or bad)  
$ git bisect reset
```



Some tips and tricks

- Cherry-picking

```
$ git cherry-pick <commit>
```

- Move this away for a while, I need to fix something first

```
$ git stash "work in progress for foo thing"
```

```
changes, commit, push and whatever
```

```
$ git stash apply
```

- Merge this, but please, do not change the history because I have some embarrassing private commits

```
$ git merge --squash <branch>
```

```
$ git commit -a
```

```
$ git push
```

- Looking for a regression

```
$ git bisect start
```

```
$ git bisect good <commit-id>
```

```
$ git bisect bad <commit-id>
```

```
$ git bisect [good|bad]
```

```
(repeat until you find the guilty commit
```

```
telling git if current version is good or bad)
```

```
$ git bisect reset
```



Some tips and tricks

- Cherry-picking

```
$ git cherry-pick <commit>
```

- Move this away for a while, I need to fix something first

```
$ git stash "work in progress for foo thing"  
changes, commit, push and whatever  
$ git stash apply
```

- Merge this, but please, do not change the history because I have some embarrassing private commits

```
$ git merge --squash <branch>  
$ git commit -a  
$ git push
```

- Looking for a regression

```
$ git bisect start  
$ git bisect good <commit-id>  
$ git bisect bad <commit-id>  
$ git bisect [good|bad]  
(repeat until you find the guilty commit  
telling git if current version is good or bad)  
$ git bisect reset
```



Patches

- Creating series of patches

```
$ git format-patch <commit-from>..<commit-to>
```

- Applying series of patches

```
$ cat patch1 patch2 .. patchn > patches.mbox
```

```
$ git am patches.mbox
```



I use Git in the dark and nobody knows (git-svn)

- Import a svn repository

```
$ git-svn clone <svn-uri> <modulename>
```

- Committing to the svn repository

```
$ git-svn dcommit # From master branch
```

- Update the git repository from the remote svn

```
$ git-svn rebase # From master branch
```



Workflow (summary)

```
$ git-svn clone <svn-uri> <modulename>
$ git checkout -b super-feature
commits and more commits
$ git checkout master
$ git-svn rebase
(if there are changes we also rebase
super-feature branch from master)
$ git merge --squash super-feature
(Do not forget to update the ChangeLog file,
remember we are committing to a svn repository now)
$ git commit -a
$ git-svn dcommit
```

