

A Practical Introduction to `git`

Emanuele Olivetti

NeuroInformatics Laboratory (NILab)
Bruno Kessler Foundation (FBK), Trento, Italy
Center for Mind and Brain Sciences (CIMEC), University of Trento, Italy
<http://nilab.fbk.eu>
olivetti@fbk.eu

Outline

- Version Control: **git**.
- Scenario 1: **single** developer, **local** repository.
 - Demo **single+local**
- Scenario 2: **Team** of developers, **central remote** repository. Minimalistic.
 - Demo **multi+remote**
- Branching.
- Scenario 3: Contributing to a Software Project hosted on **GitHub**.
- Extras: how to set up centralised repository, and more.

Version Control: Naming & Meaning

Wikipedia

“Revision control, also known as version control, source control or software configuration management (SCM), is the management of changes to documents, programs, and other information stored as computer files.”

Popular Acronyms:

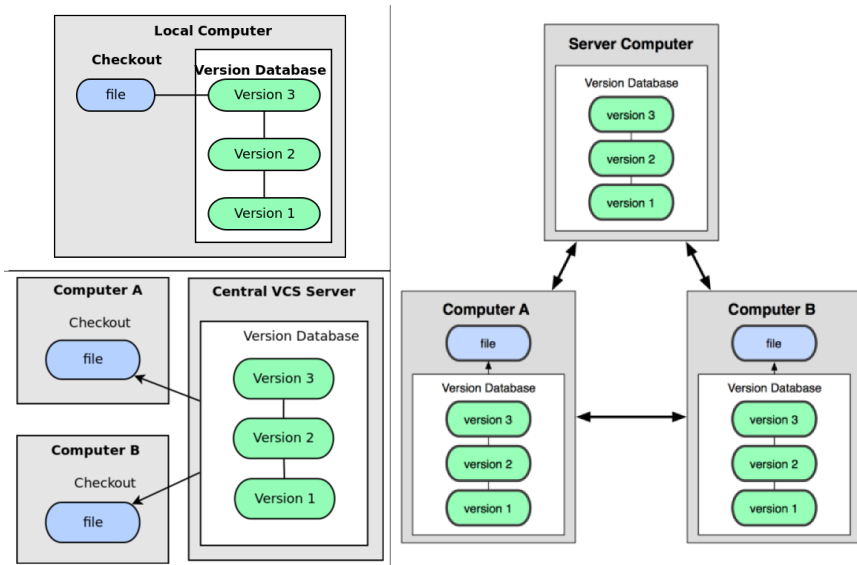
- VC
- SCM

Misnomer:

- Versioning

Q: have you ever used VC? (raise your hand = YES)

Version Control: Local, Centralized, Distributed



From *Pro Git*, S.Chacon 2009, CC 3.0 license.

Survey: `git`

- Q1: Have you heard about `git`?
- Q2: Do you use `git`?
- Q3: Why the “`git`” name? (from `git` FAQ)
 - 1 Random three-letter combination that is pronounceable.
 - 2 Acronym (global information tracker).
 - 3 Irony.

git? Why “git”?

Linus Torvalds: “I name all my projects after myself. First Linux, now **git**.”



<http://www.merriam-webster.com/dictionary/git>

git  *noun* \ˈgit\

Definition of GIT

British : a foolish or worthless person

Examples of GIT

- That *git* of a brother of yours has ruined everything!
- <oh, don't be such a silly *git*, of course your mates want you around>

Origin of GIT

variant of *get*, term of abuse, from ²*get*

First Known Use: 1929

Related to GIT

Synonyms: *berk* [*British*], *booby*, *charlie* (also *charley*) [*British*], *cuckoo*, *ding-a-ling*, *dingbat*, *ding-dong*, *dipstick*, *doofus* [*slang*], *featherhead*, *fool* [*British*], *goose*, *half-wit*, *jackass*, *lunatic*, *mooncalf*, *nincompoop*, *ninny*, *ninnvhammer*, *nit* [*chiefly British*], *nitwit*, *nut*, *nutcase*, *simp*.

git

usage: git [OPTIONS] COMMAND [ARGS]

The most commonly used git commands are:

| | |
|--------|--------------------------------------|
| add | Add file contents to the index |
| commit | Record changes to the repository |
| diff | Show changes between commits, commit |
| ... | |

git help <command>

git status

Introduce yourself to **git**:

```
git config --global user.name "Emanuele Olivetti"
```

```
git config --global user.email "olivetti@fbk.eu"
```


git. Single developer + local repository.

Scenario 1: single developer + local repository.

Single+Local `git`. Motivations.

- **Q:** do you use VC for local repo?
- Why VC for single developer + local repository?
 - First step towards a shared project.
 - Backup.
 - To keep the memory of your work.

Single+Local `git`. Init.

`git init`

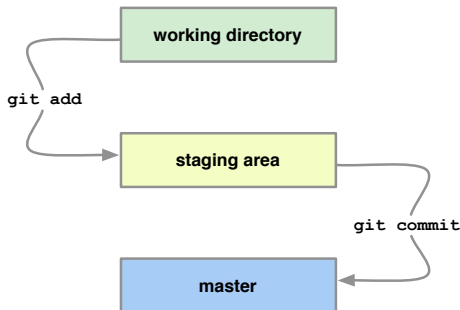
- Creates an empty `git` repository.
- Creates the git directory: `.git/`



Note: it is **safe**. It does not change your pre-existing files.

Single+Local `git`. The tracking process.

```
git add <filename>
```



```
git commit -m "Let us begin."
```

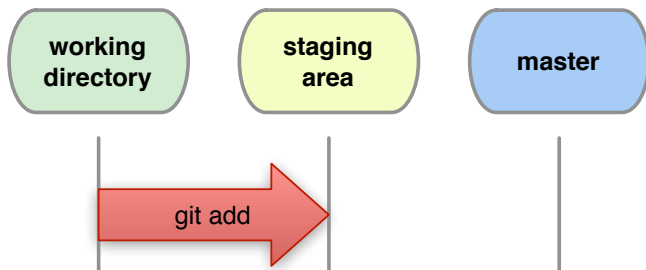
Wikipedia

“A *staging area* is a location where organisms, people, vehicles, equipment or material are assembled before use”.

Single+Local **git**. Add.

```
git add file1 [file2 ...]
```

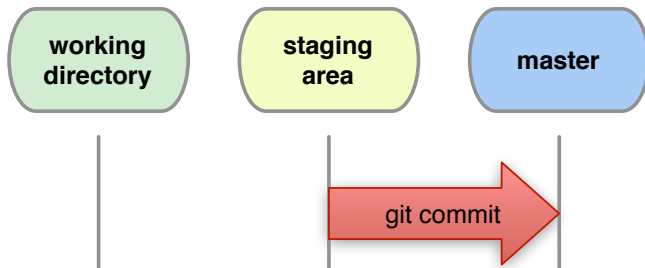
- Adds new files for next commit.
- Adds content from working dir to the staging area (index) for next commit.
- DOES NOT add info on file permissions other than *exec/noexec* (755 / 644).
- DOES not add directories *per se*.



Single+Local `git`. Commit.

```
git commit [-m "Commit message."]
```

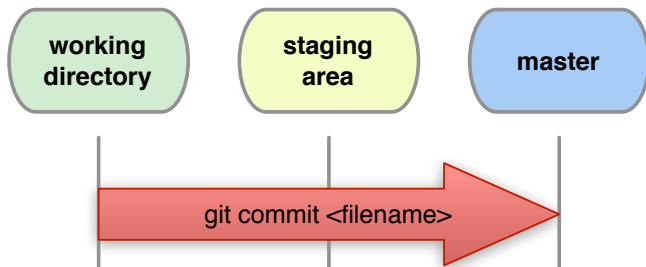
Records changes from the staging area to master.



Single+Local `git`. Commit.

```
git commit file1 file2
```

Records all changes of `file1`, `file2` from working dir and staging area to master.



```
git commit -a
```

Records all changes in working dir and staging area. *Be Careful!*

Single+Local **git**. Commit names. OPTIONAL

- Every *commit* is a **git**-object.
- The history of a project is a graph of objects referenced by a 40-digit **git**-name: *SHA1(object)*.
- *SHA1(object)* = 160-bit Secure Hash Algorithm.
- Examples:

```
$ git commit README -m "Added README."
```

```
[master dbb4929] Added README.
```

```
1 files changed, 1 insertions(+), ...
```

or

```
$ git log
```

```
commit dbb49293790b84f0bdcd74fd9fa5cab0...
```

```
Author: Emanuele Olivetti <olivetti@fbk.eu>
```

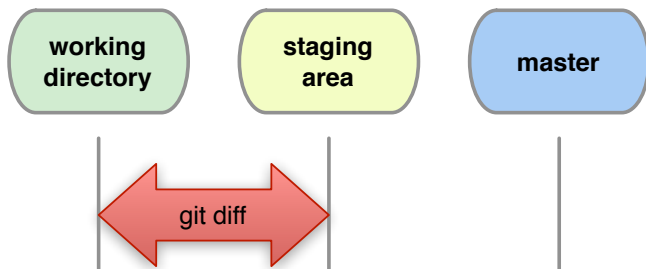
```
Date:   Wed Sep 15 00:08:46 2010 +0200
```

```
...
```


Single+Local `git`. Diff.

`git diff`

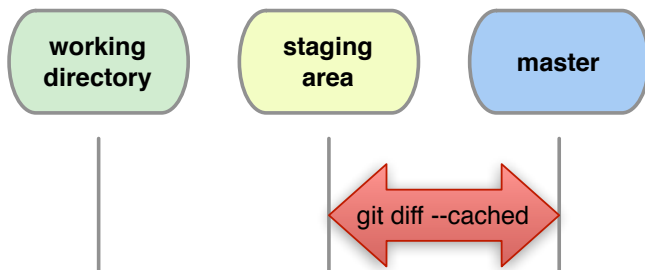
Shows what changes between *working directory* and *staging area (index)*.



Single+Local **git**. Diff. OPTIONAL

Q: “**git add**” then “**git diff**”. What output?

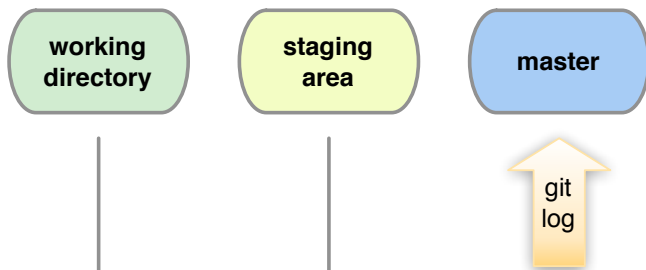
git diff --cached shows differences between index and last commit (**HEAD**).



Single+Local `git`. Logs.

`git log`

Shows details of the commits.



Single+Local git. Logs.

gitk

GUI to browse the git repository.

The screenshot shows the gitk graphical user interface. At the top, a commit log is displayed with a graph of commits on the left. The log entries include merge operations and patches for USB, PCI Hotplug, and ARM. Below the log, the SHA1 ID of the selected commit is shown as 9f793d2c77ec5818679e4747c554d9333cec4f76. The main pane shows the commit details for the selected commit, including the author (Pete Zaitcev), committer (Greg Kroah-Hartman), and the commit message. The message describes a patch for USB: fix ub issues, which smoothes two imperfections: increasing the number of LUNs per device and replacing mdelay with msleep. The file browser on the right shows the file drivers/block/ub.c.

merge rsync://rsync.kernel.org/pub/scm/linux/kernel/git/c...
Merge rsync://rsync.kernel.org/pub/scm/linux/kernel/git/c...
[PATCH] USB: fdi_sio: avoid losing received data in tty-l...
[PATCH] USB: fix ub issues
[PATCH] PCI Hotplug: fix CPCI reference counting bug
[IA64] Fix race condition in the rt_sigprocmask fastcall
Merge master.kernel.org:/home/rmk/linux-2.6-arm
[PATCH] sg traverse fix for __ata_pi_bytes()
[PATCH] sata_sil: Fix FIFO PCI Bus Arbitration kernel oo...
[PATCH] ARM: Remove zero-byte sized file
Merge rsync://rsync.kernel.org/pub/scm/linux/kernel/git/daven...
[PKT_SCHED]: Fix numeric comparison in meta ematch

Linus Torvalds <torvalds@ppc970.osdl.org>
Linus Torvalds <torvalds@ppc970.osdl.org>
Ian Abbott <abbotti@mev.co.uk>
Pete Zaitcev <zaitcev@redhat.com>
Scott Murray <scottm@somantnetworks.com>
Christoph Lameter <clameter@sgi.com>
Linus Torvalds <torvalds@ppc970.osdl.org>
Albert Lee <albertcc@tw.ibm.com>
Jens Axboe <axboe@suse.de>
Russell King <rmk@dyn-67.arm.linux.org.uk>
Linus Torvalds <torvalds@ppc970.osdl.org>
Thomas Graf <tgraf@suug.ch>

SHA1 ID: 9f793d2c77ec5818679e4747c554d9333cec4f76 Find Ex

Author: Pete Zaitcev <zaitcev@redhat.com> 2005-06-06 14:54:59
Committer: Greg Kroah-Hartman <gregkh@suse.de> 2005-06-09 02:38:11

[PATCH] USB: fix ub issues

This smoothes two imperfections:
- Increase number of LUNs per device from 4 to 9. The best solution
would be to remove this limit altogether, but that has to wait until
the time when more than 26 hosts are allowed.
- Replace mdelay with msleep in a probing routine.

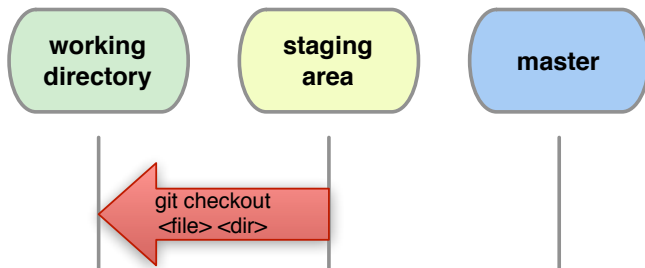
Signed-off-by: Pete Zaitcev <zaitcev@yahoo.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

All files
drivers/block/ub.c

Single+Local `git`. “How to clean this mess??” OPT.

```
git checkout <filename>
```

Get rid of what changed in `<filename>` (between working dir and staging area).



Single+Local `git`. Time travelling. OPTIONAL

Back to the past when you did commit `dbb49293790b84...`

```
git checkout dbb4929
```

...and now, *back to the present!*

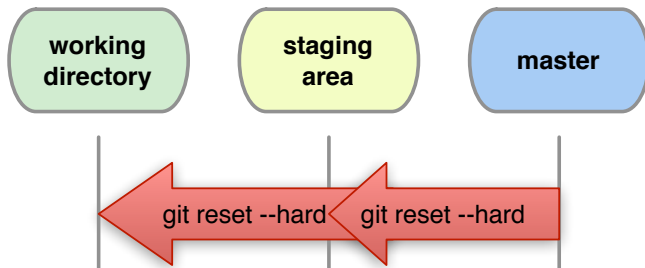
```
git checkout master
```

Single+Local `git`. “How to clean this mess??”. OPT.

First read *carefully* `git status`. If you panic:

```
git reset --hard HEAD
```

Restore all files as in the last commit.



Warning: `reset` can destroy history!

Single+Local **git**. (Re)move. OPTIONAL

Warning: whenever you want to *remove*, *move* or *rename* a tracked file use **git**:

```
git rm <filename>
```

```
git mv <oldname> <newname>
```

Remember to **commit** these changes!

```
git commit -m "File (re)moved."
```

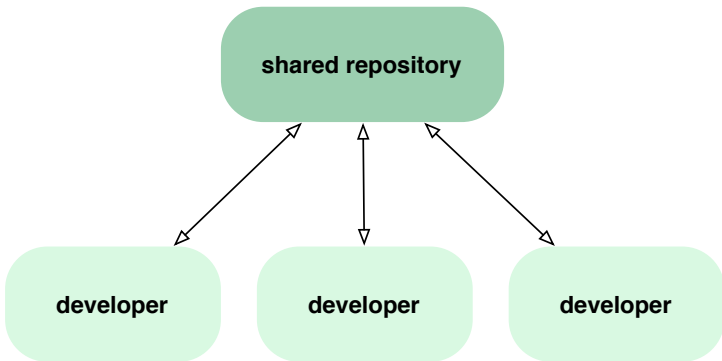

Single+Local **git**. Demo.

Demo: `demo_git_single_local.txt`

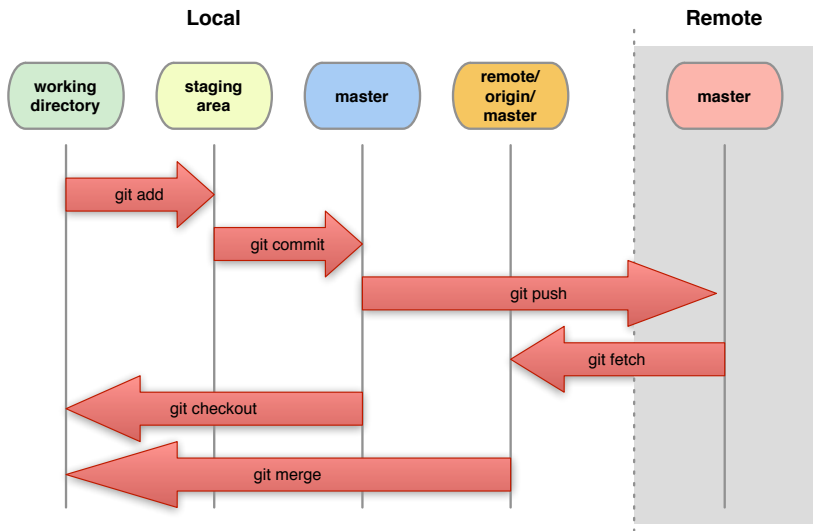
multi+remote/shared **git**.

Scenario 2: multiple developers + remote central repository.

multi+remote/shared **git**.



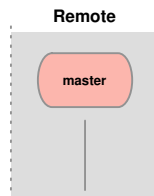
multi+remote/shared git.



multi+remote/shared git.

```
git clone <URL>
```

Creates *two* local copies of the *whole* remote repository.



Available transport protocols:

- `ssh://`, `git://`, `http://`, `https://`, `file://`

Ex.: `git clone https://github.com/ASPP/pelita.git`

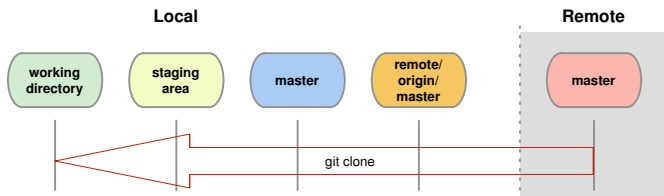
```
git remote -v
```

shows **name** and **URL** of the remote repository.

multi+remote/shared git.

```
git clone <URL>
```

Creates *two* local copies of the *whole* remote repository.



Available transport protocols:

■ `ssh://`, `git://`, `http://`, `https://`, `file://`

Ex.: `git clone https://github.com/ASPP/pelita.git`

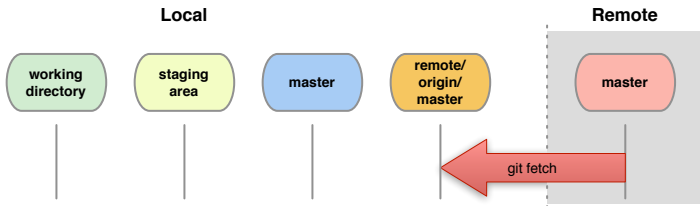
```
git remote -v
```

shows **name** and **URL** of the remote repository.

multi+remote/shared git. Fetch.

git fetch

- Downloads updates from remote master to local remote master.
- The local master, staging area and working directory do not change.



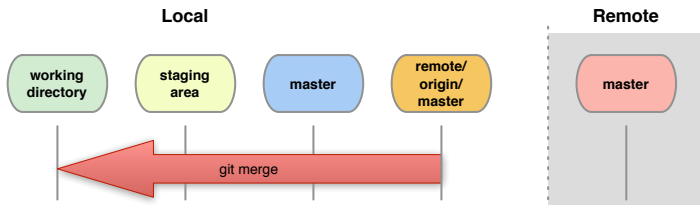
Q: Why **origin**?

A: Just a label for **Remote**. Choose the one you like.

multi+remote/shared git. Merge.

git merge

- Joins development histories together.
- **Warning**: can generate *conflicts*!
- **Note**: it merges only when all changes are committed.



`git fetch + git merge = git pull`

Conflict!

```
...  
<<<<<< yours:sample.txt  
Conflict resolution is hard;  
let's go shopping.  
=====  
Git makes conflict resolution easy.  
>>>>>> theirs:sample.txt  
...
```

multi+remote/shared **git**. Conflicts.

How to resolve conflicts.

- 1 See where conflicts are:

```
git diff
```

- 2 Edit conflicting lines.

- 3 Add changes to the staging area:

```
git add file1 [...]
```

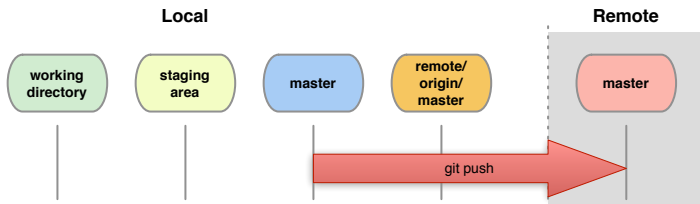
- 4 Commit changes:

```
git commit -m "Conflicts solved."
```

multi+remote/shared **git**.

`git push`

- Updates *remote masters* (both Local and Remote).
- Requires **fetch+merge** first.



Demo: `demo_git_multi_remote.txt`.

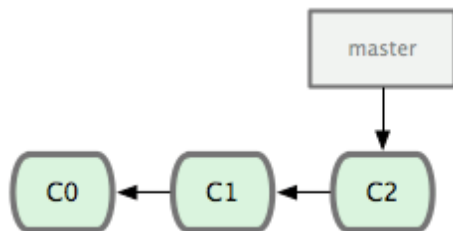
Other related files:

- `create_remote_repo_sn.sh`
- `collaborator1.sh`
- `collaborator2.sh`
- `collaborator2.sh`

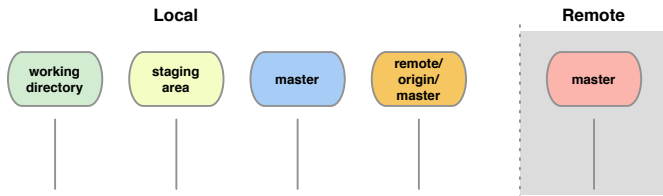
Branching.

basic branching

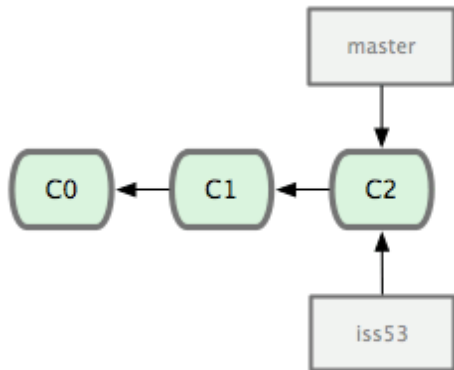
Branching.



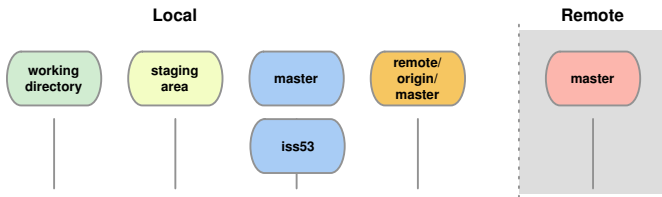
```
git commit (C0)
git commit (C1)
git commit (C2)
```



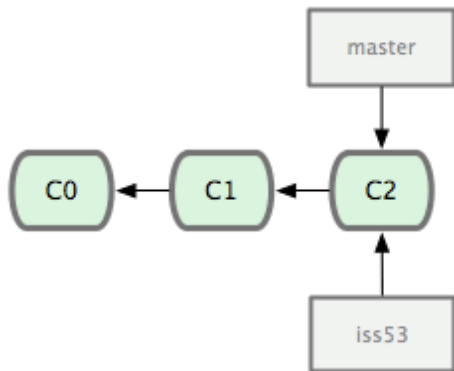
Branching.



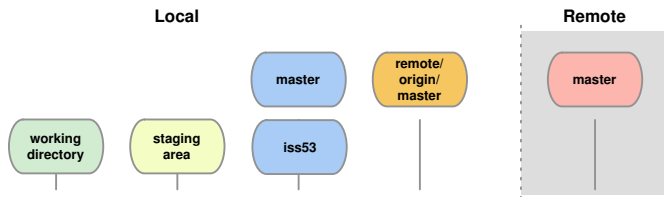
git branch iss53
git checkout iss53
git checkout master



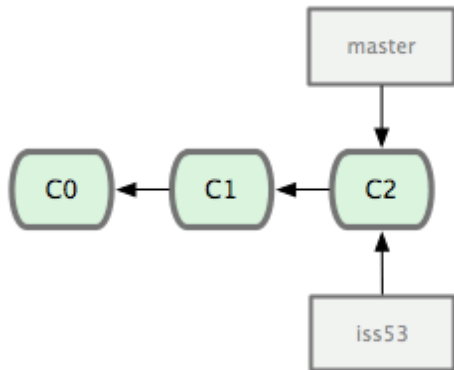
Branching.



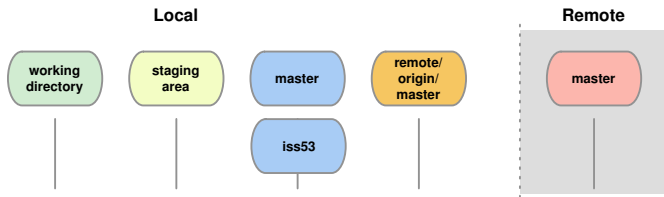
git branch iss53
git checkout iss53
git checkout master



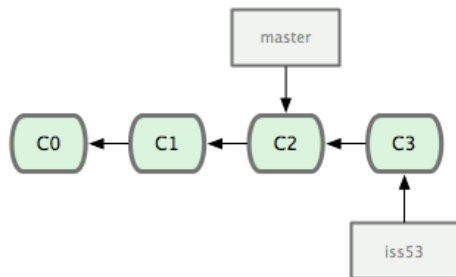
Branching.



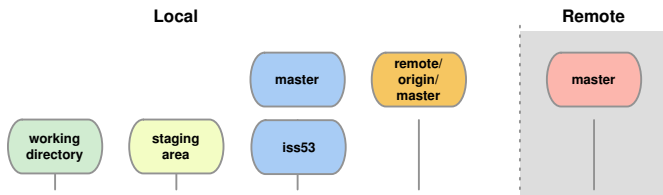
git branch iss53
git checkout iss53
git checkout master



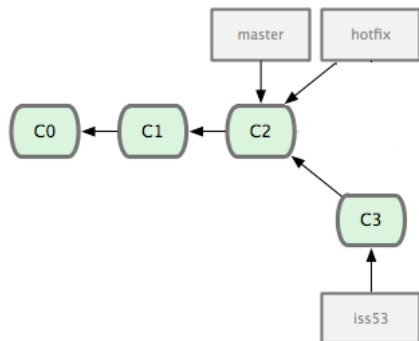
Branching.



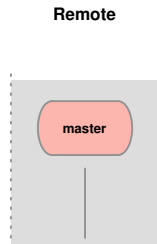
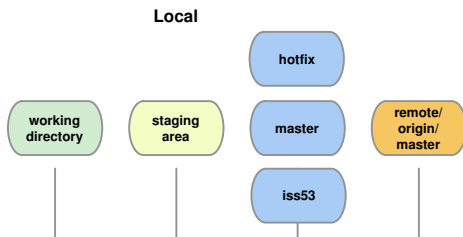
```
git checkout iss53  
git commit (C3)
```



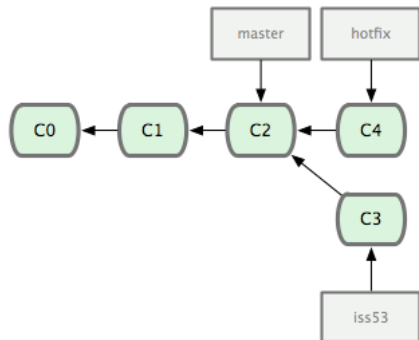
Branching.



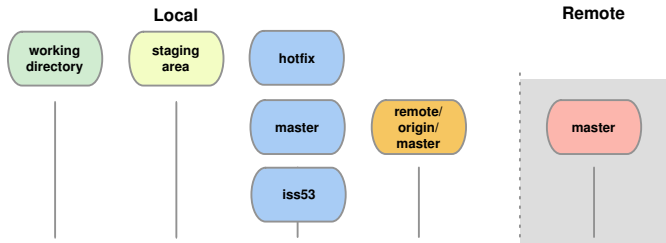
git branch hotfix
git checkout hotfix
git commit (C4)



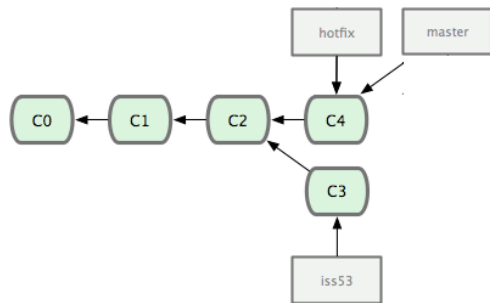
Branching.



git branch hotfix
git checkout hotfix
git commit (C4)

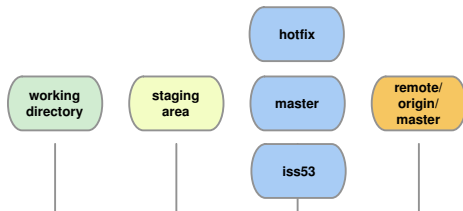


Branching.

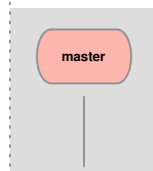


```
git checkout master  
git merge hotfix
```

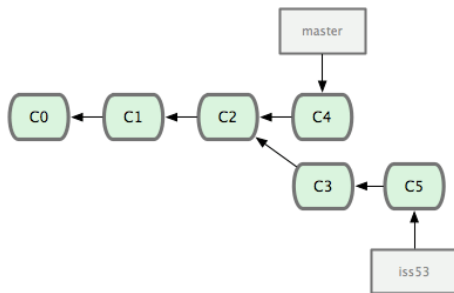
Local



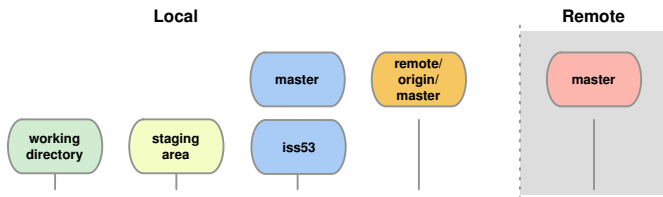
Remote



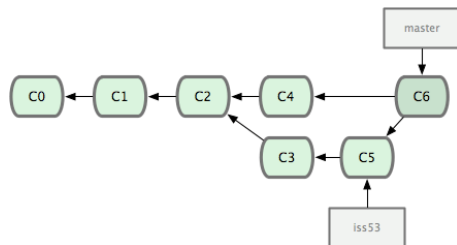
Branching.



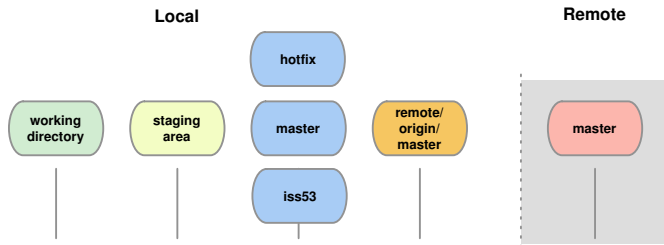
```
git checkout iss53  
git commit (C5)
```



Branching.



```
git checkout master  
git merge iss53
```

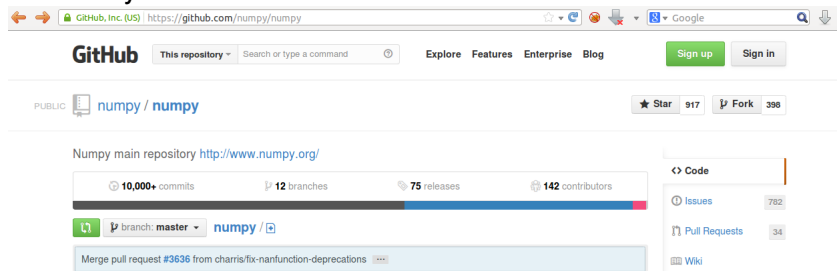


Contributing through GitHub

Scenario 3: contributing to a software project hosted on GitHub.

Contributing through GitHub

Q: Have you ever heard of GitHub?



The screenshot shows the GitHub web interface for the 'numpy/numpy' repository. At the top, the GitHub logo is on the left, and navigation links for 'Explore', 'Features', 'Enterprise', and 'Blog' are on the right. Below the navigation bar, the repository name 'numpy / numpy' is displayed with a 'PUBLIC' label. To the right of the repository name are buttons for 'Star' (917) and 'Fork' (398). Below this, the text 'Numpy main repository http://www.numpy.org/' is shown. A progress bar indicates repository statistics: 10,000+ commits, 12 branches, 75 releases, and 142 contributors. Below the progress bar, there is a 'branch: master' dropdown and a 'Merge pull request #3636 from charris/fix-nanfunction-deprecations' button. On the right side of the repository page, there are links for 'Code', 'Issues' (782), 'Pull Requests' (34), and 'Wiki'.

What is GitHub?

- Wikipedia: *"GitHub is a web-based hosting service for software development projects that use **git**".*
- 5 millions repositories (Jan 2013).
- Commercial...
- ...but friendly to Free / Open Source software projects.

Contributing through GitHub

Assumptions

- You use a software and feel ready to contribute to it.
- The software project is hosted on `http://github.com`

Intuitive Idea

- You **do not push** your changes to the main repository.
- Instead you create a public copy (**fork**) of the main repository...
- ...and then push your changes to that.
- Then you ask the owners of the main repository if they like your changes and want to merge them (**pull request**).

Contributing through GitHub. Not for everyone ;-)



torvalds commented

I don't do github pull requests.

github throws away all the relevant information, like having even a valid email address for the person asking me to pull. The diffstat is also deficient and useless.

Git comes with a nice pull-request generation module, but github instead decided to replace it with their own totally inferior version. As a result, I consider github useless for these kinds of things. It's fine for *hosting*, but the pull requests and the online commit editing, are just pure garbage.

I've told github people about my concerns, they didn't think they mattered, so I gave up. Feel free to make a bugreport to github.

Linus

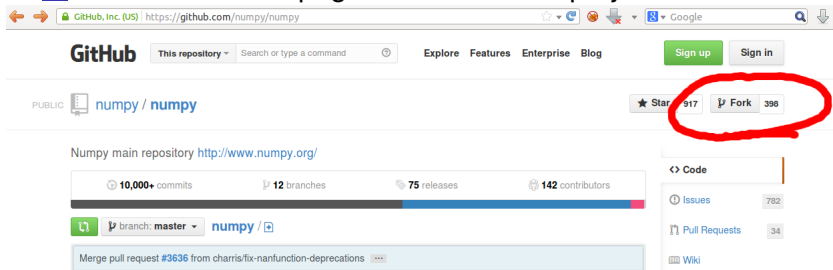
...

<https://github.com/torvalds/linux/pull/17>

Contributing through GitHub: Recipe I

1 **Register** on <http://github.com>

2 **Visit** the GitHub page of the software project and **Fork** it:



3 **Clone your copy** of the project on your computer.

```
git clone git@github.com:<login>/<project>.git
```

4 Create a **branch** to host your improvements.

```
■ git branch <new-feature>
```

```
■ git checkout <new-feature>
```



Contributing through GitHub: Recipe II


5 Add your improvements.

- `git add <new-file>`
- `git commit -m ...`

6 **Push** your improvements.

```
git push origin <new-feature>
```

7 Send a **pull request**.


 [Compare & pull request](#)

Write


Preview

Comments are parsed with [GitHub Flavored Markdown](#)

Leave a comment



Attach images by dragging & dropping, [selecting them](#), or pasting from the clipboard.



✓ **Able to merge**

These branches can be automatically merged

[Send pull request](#)

Detailed Explanation

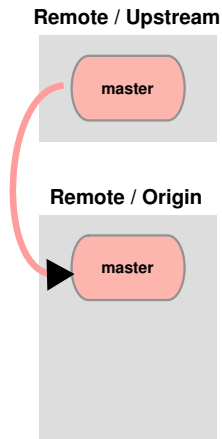
Contributing through GitHub: Detailed Explanation

Remote / Upstream



There is a software project hosted on remote GitHub repository (**upstream**). You want to improve it.

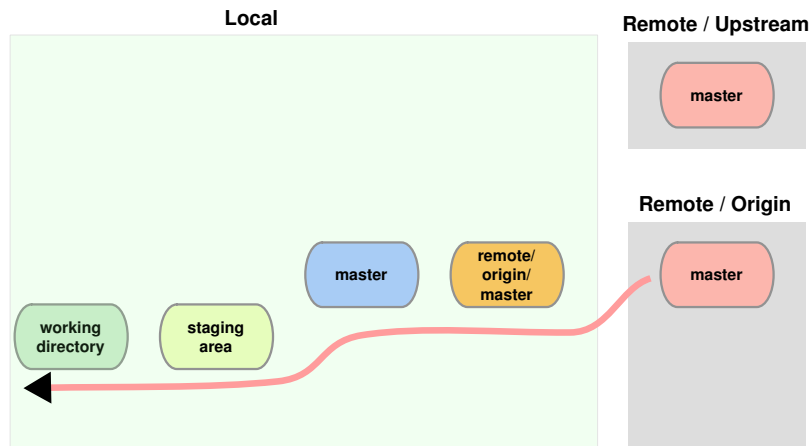
Contributing through GitHub: Detailed Explanation



So you **fork** it by creating a (remote) copy of it:

```
git clone --bare <UPSTREAM_URL>
```

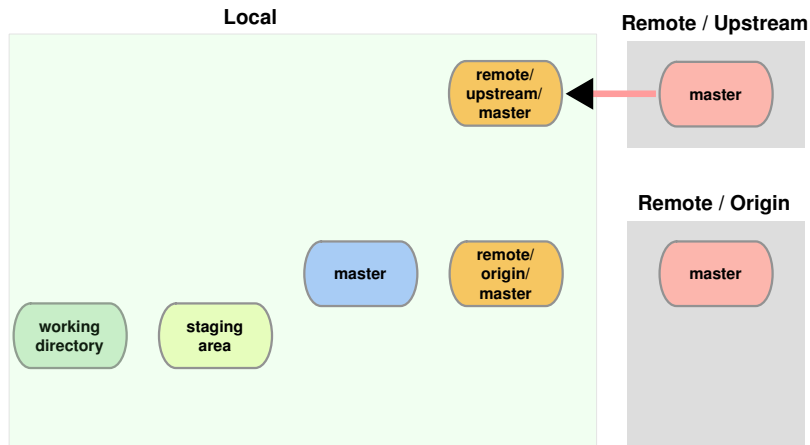

Contributing through GitHub: Detailed Explanation



Now you clone your copy on your local computer:

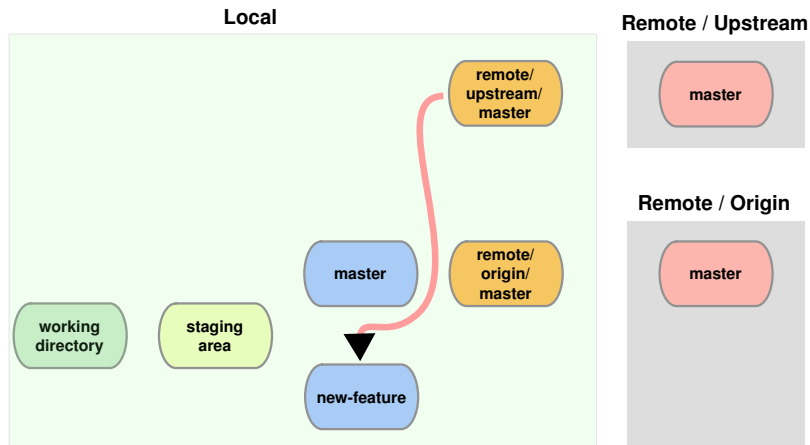
```
git clone <ORIGIN_URL>
```

Contributing through GitHub: Detailed Explanation



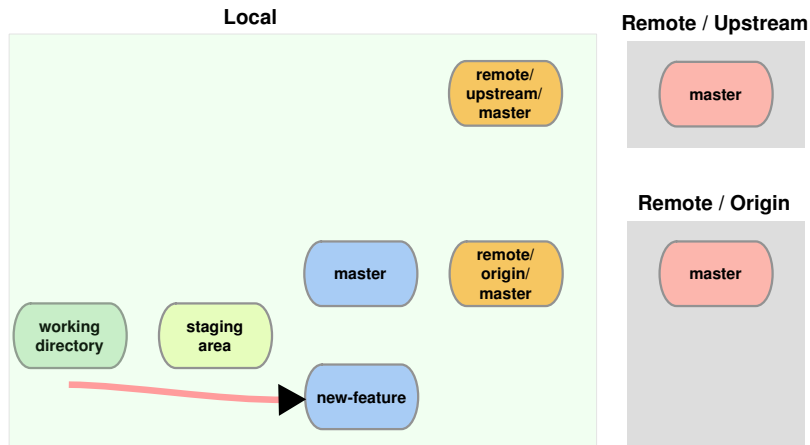
```
git remote add upstream <UPSTREAM_URL>
git fetch upstream
```

Contributing through GitHub: Detailed Explanation



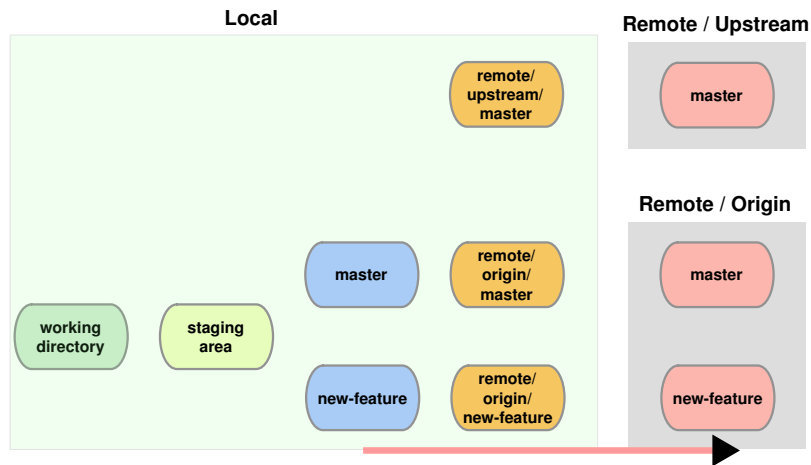
```
git branch new-feature upstream/master
git checkout new-feature
```

Contributing through GitHub: Detailed Explanation



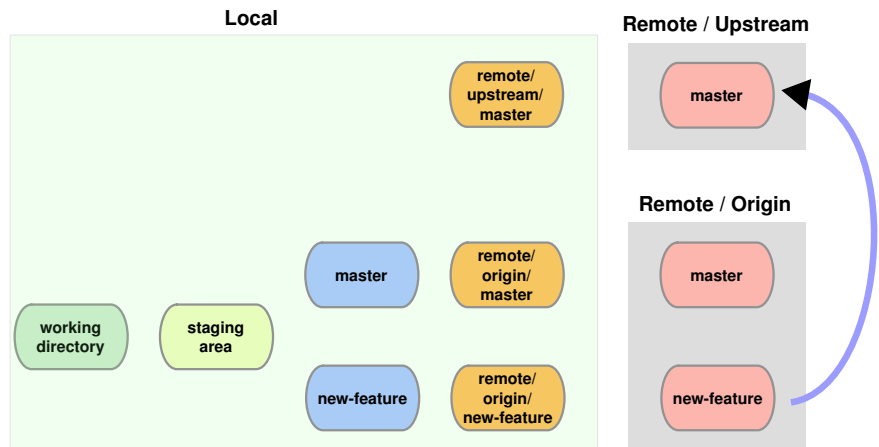
```
git add ...  
git commit ...
```

Contributing through GitHub: Detailed Explanation



publish your new feature:
`git push origin new-feature`

Contributing through GitHub: Detailed Explanation



Notify the owners of the main repository about **new-feature**
they: `git fetch` + (eventually) `git merge`

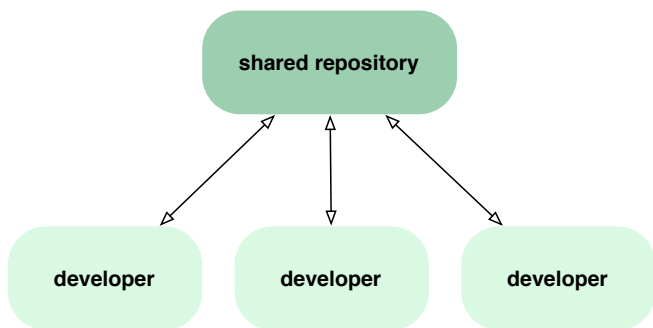
Setting up a remote+shared repository. OPTIONAL

GOAL: I want to share my local repository so others can **push**.

“Why can't I just *extend permissions* in my **local** repo?”

- Yes you can...
- ...but your colleagues will not push (**read-only**).

To have it **read-write**: set up a **remote** *shared* repository.



Setting up a remote+shared repository. OPTIONAL

You have a local repository and want to share it (**ssh**) from a remote server on which your colleagues already have access.

On *remote* server create **bare+shared** repository:

- `mkdir newproject`
- set up proper *group* permissions: `chmod g+rws newproject`
- `cd newproject`
- `git --bare init --shared=group`

On *local* machine push your repository to remote:

- `git remote add origin
ssh://remote.com/path/newproject`
- `git push -u origin master`

Setting up a remote+shared repository. OPTIONAL

Demo: `demo_git_setup_remote.txt`.

- **Rike-Benjamin Schuppner**
- Zbigniew Jędrzejewski-Szmek
- Tiziano Zito
- Bastian Venthur
- `http://progit.com`
- `apcmag.com`
- `lwn.net`
- `http://www.markus-gattol.name/ws/scm.html`
- `http://matthew-brett.github.io/pydagogue/
gitwash/git_development.html`

I want to know more about **git**!

Understanding how **git** works:

- **git** foundations, by Matthew Brett:
<http://matthew-brett.github.com/pydagogue/foundation.html>
- The **git** parable, by Tom Preston-Werner:
<http://tom.preston-werner.com/2009/05/19/the-git-parable.html>

Excellent guides:

- “Pro Git” book: <http://git-scm.com/book> (FREE)
- **git** magic: <http://www-cs-students.stanford.edu/~blynn/gitmagic/>

Contributing to a project hosted on GitHub:

- “Gitwash”, by Matthew Brett:
http://matthew-brett.github.io/pydagogue/gitwash/git_development.html

Gource:

<http://code.google.com/p/gource/>

Copyright Emanuele Olivetti, 2014

This presentation is distributed under the license

Creative Commons *Attribution* 3.0

<https://creativecommons.org/licenses/by/3.0/>

The diagrams of the branching example are taken from *Pro Git*, (copyright S.Chacon, 2009) and are distributed under the license Creative Commons 3.0 Attribution-Non Commercial-Share Alike.