Formation Git/GitHub

Fabien Engels

Table of Contents

1. Introduction

1.1. Tracking your code

1.2. Why using over SVN ?

- Branches are cheap
- Fast, store whole files not differences
- Not centralized (DVCS vs VCS)
- Work offline (commit offline and push when online)

1.3. Sharing your code

Table 1. Some projects hosted by GitHub

Project name	Collaborators	Commits	Home
Python	182	> 98k	https://github.com/python/cpython
Linux	∞	> 662k	https://github.com/torvalds/linux
Django	1377	> 24k	https://github.com/django/django
Homebrew	6307	> 79k	https://github.com/Homebrew/homebrew-core

Table 2. Seismology related projects

Project name	Collaborators	Commits	Home
Obspy	63	> 10k	https://github.com/obspy/obspy
SeisComP3	20	> 500	https://github.com/SeisComP3/seiscomp3
MSNoise	4	> 400	https://github.com/ROBelgium/MSNoise
libmseed	3	> 110	https://github.com/iris-edu/libmseed
ringserver	2	> 118	https://github.com/iris-edu/ringserver
waveloc	1	> 560	https://github.com/amaggi/waveloc
wphase	3	> 246	https://github.com/eost/wphase
seedscan	6	> 908	https://github.com/usgs/seedscan
StationXML	3	> 10	https://github.com/FDSN/StationXML

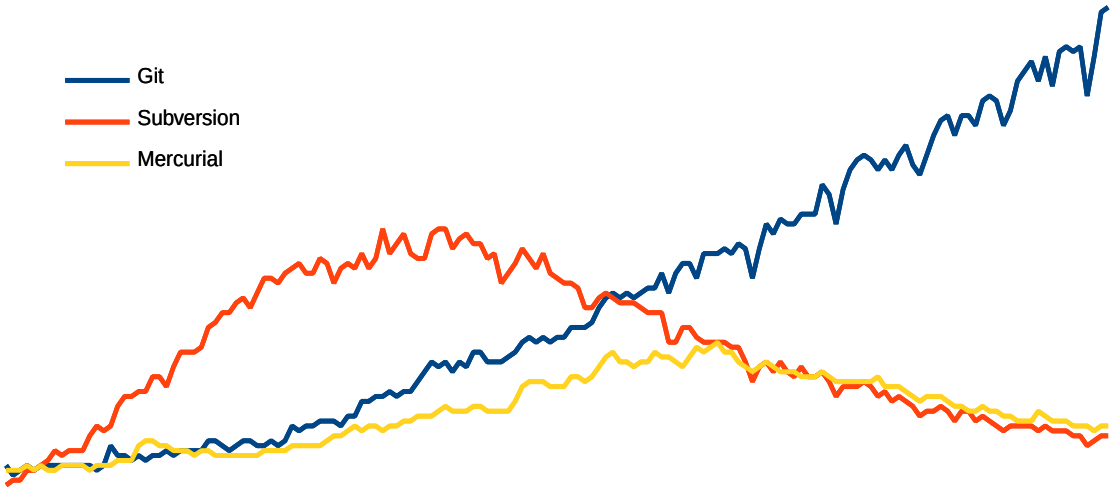
Some stats from 2016, GitHub had :

- 5.8M active users
- 331k organizations
- 19.4M active repositories
- 10.7M active issues

1.4. A bit of history

- Created by Linus Torvalds
- … to replace BitKeeper
- Released on 11th July 2005

Google searches from 2005 to 2017



2. Git, command line tool

2.1. Let’s create our first repository

Simply execute this command :

```
fabien@izanagi ~ » git init my_awesome_project
Initialized empty Git repository in /home/fabien/my_awesome_project/.git/
```

You’ve just create a directory named my_awesome_project containing an another folder named .git :

```
fabien@izanagi ~ » cd my_awesome_project
fabien@izanagi ~/my_awesome_project » tree -a
.
└──── .git
    ├──── branches
    ├──── config
    ├──── description
    ├──── HEAD
    ├──── hooks
    │    ├──── applypatch-msg.sample
    │    ├──── ...
    │    └──── update.sample
    ├──── info
    │    └──── exclude
    ├──── objects
    │    ├──── info
    │    └──── pack
    └──── refs
        ├──── heads
        └──── tags

10 directories, 14 files
```

We’ll see later what files are in the .git directory, just know git will store everything it needs in this folder. We can check the state of our repository by executing `git status` :

```
fabien@izanagi ~/my_awesome_project » git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

2.2. Our first commit

Create a file `app.py` with the most useful code :

```
#!/usr/bin/env python2

def greetings():
    print "Hello world!"


if __name__ == "__main__":
    greetings()
```

Let's check the state of the repository again :

```
fabien@izanagi ~/my_awesome_project » git status
On branch master

Initial commit

Untracked files: ①
  (use "git add <file>..." to include in what will be committed)

  app.py ①

nothing added to commit but untracked files present (use "git add" to track) ②
```

① Currently, git don't care about our code as `app.py` has the status `Untracked`.

② Generally, git suggests some action to take.

Currently, the file is not part of the repository. You need to tell git to take care of your code :

```
fabien@izanagi ~/my_awesome_project » git add app.py
```

Is your file committed ? Hum ... no yet

```
fabien@izanagi ~/my_awesome_project » git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

  new file:   app.py
```

Our file is staged. It means it'll part of our next commit. Before committing, it's always a good idea to check repository state using `git status`.

Finally, our first commit ... or not.

```
fabien@izanagi ~/my_awesome_project » git commit -m 'My first commit!'

*** Please tell me who you are. ①

Run

  git config --global user.email "you@example.com" ①
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: empty ident name (for <(null)>) not allowed
```

① The author name and email are part of the commit. Git requires them and suggests one more time action to take to solve the current issue.

Telling who you are can it be done in multiple ways :

By command lines

```
fabien@izanagi ~/my_awesome_project » git config --global user.email
"fabien.engels@unistra.fr"
fabien@izanagi ~/my_awesome_project » git config --global user.name "Fabien Engels"
```

Or directly creating a config file named `~/.gitconfig` :

```
[user]
  email = fabien.engels@unistra.fr
  name = Fabien Engels
```

Let's try to commit again

```
fabien@izanagi ~/my_awesome_project » git commit -m 'My first commit!'
[master (root-commit) 64188b7] My first commit!
 1 file changed, 8 insertions(+)
 create mode 100755 app.py
```

It worked, you've done your first commit !

```
fabien@izanagi ~/my_awesome_project » git log
commit 64188b75074257c639920a2a45e00130aea7219f
Author: Fabien Engels <fabien.engels@unistra.fr>
Date:   Tue Feb 28 09:34:14 2017 +0100

    My first commit!
```

2.3. A first look to .git directory

Our commit created multiple object files inside .git directory.

```
fabien@izanagi ~/my_awesome_project » find .git/objects -type f
.git/objects/88/cdc3534dd04abe83564c4f4dd4ac8e5b0d41de
.git/objects/64/188b75074257c639920a2a45e00130aea7219f ①
.git/objects/a6/1f1b413ea15b1dc692cc7e55b6f060edf268e3
```

① Our commit

Display object corresponding to our commit

```
fabien@izanagi ~/my_awesome_project » git cat-file -p 64188b
tree a61f1b413ea15b1dc692cc7e55b6f060edf268e3 ①
author Fabien Engels <fabien.engels@unistra.fr> 1488270854 +0100
committer Fabien Engels <fabien.engels@unistra.fr> 1488270854 +0100

My first commit!
```

① Reference to another object !

What inside in this other object

```
fabien@izanagi ~/my_awesome_project » git cat-file -p a61f1b
100755 blob 88cdc3534dd04abe83564c4f4dd4ac8e5b0d41de  app.py ①
```

① Reference to a blob object !

Have a look to this blob object … it's our code !

```
fabien@izanagi ~/my_awesome_project » git cat-file -p 88cdc3
#!/usr/bin/env python2

def greetings():
    print "Hello world!"


if __name__ == "__main__":
    greetings()
```

You can shorten IDs until there is no ambiguity. ex: ID a61f1b413ea15b1dc692cc7e55b6f060edf268e3 can be shorten as a61f1b

Git store snapshots of your code, not differences like SVN. It's one of the reason Git is so fast (but use more space).

2.4. Improve our code

Have a look to this blob object ... it's our code !

```
#!/usr/bin/env python2

def greetings():
    print "Hello RESIF people!"


if __name__ == "__main__":
    greetings()
```

You can visualize your current modifications between your workdir and repository head :

```
fabien@izanagi Sync/my_awesome_project » git diff
diff --git a/app.py b/app.py
index 88cdc35..e35169d 100755
--- a/app.py
+++ b/app.py
@@ -1,7 +1,7 @@
 #!/usr/bin/env python2

 def greetings():
-    print "Hello world!"
+    print "Hello RESIF people!"


 if __name__ == "__main__":
```

Of course in order to commit your change, you need to stage your file and then commit it

```
fabien@izanagi ~/my_awesome_project » git add app.py
fabien@izanagi ~/my_awesome_project » git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  modified:   app.py
fabien@izanagi ~/my_awesome_project » git commit -m 'What an improvement!'
[master b6cb2cc] What an improvement!
 1 file changed, 1 insertion(+), 1 deletion(-)
fabien@izanagi ~/my_awesome_project (master) » git log
commit b6cb2cc4a5034ae5a5bf35830cdec761cb2d6f1d
Author: Fabien Engels <fabien.engels@gmail.com>
Date:   Thu Mar 2 14:45:25 2017 +0100

    What an improvement!

commit 64188b75074257c639920a2a45e00130aea7219f
Author: Fabien Engels <fabien.engels@unistra.fr>
Date:   Tue Feb 28 09:34:14 2017 +0100

    My first commit!
```

2.5. Branches

Branches is a powerful feature of Git. It allows to "fork" your code and to isolate the code the time you develop a geature or fix a bug. Any code belongs to a branch even the one you've just created as Git creates a default branch called "master".

You can list the branches of your repository with the following command

```
fabien@izanagi ~/my_awesome_project » git branch
* master
```

Creating a branch is very cheap and easy

```
fabien@izanagi ~/my_awesome_project » git branch python3
fabien@izanagi ~/my_awesome_project » git branch
* master ①
  python3
```

① The asterisk indicates your current branch

To work on your new branch, you need to checkout it

```
fabien@izanagi ~/my_awesome_project » git checkout python3
Switched to branch 'python3'
fabien@izanagi ~/my_awesome_project » git status
On branch python3
nothing to commit, working tree clean
```

You can create and checkout a branch in one command : `git checkout -b python3`

Now we have a nice "python3" branch, it's time to update our code to bring Python3 compability

```
#!/usr/bin/env python

def greetings():
    print("Hello RESIF people!")


if __name__ == "__main__":
    greetings()
```

And to commit our modifications

```
fabien@izanagi ~/my_awesome_project » git commit -a -m 'Add Python3 support'
[python3 34350f2] Add Python3 support
 1 file changed, 2 insertions(+), 2 deletions(-)
```

You can use `-a` flag to automatically stage all the modifications while you commit

Let's have a look to the repository graph

```
                                                                    ③
    * 34350f2 - Add Python3 support                          (HEAD -> python3)
    |                                                                ②
    |
    |
    * b6cb2cc - What an improvement!                                (master)
    |                                                                ①
    |
    |
    * 64188b7 - My first commit!
```

① Our previous was made on the `master` branch...

② ... while the last was make on the `python3` branch

③ Our last commit became the new HEAD of the repository

Suddlendly, we need to bring some modifications on our main branch

```
fabien@izanagi ~/my_awesome_project » git checkout master
Switched to branch 'master'
fabien@izanagi ~/my_awesome_project » cat app.py ①
#!/usr/bin/env python2

def greetings():
    print "Hello RESIF people!"


if __name__ == "__main__":
    greetings()
```

① Git has updated automatically our workdir with the last version of the code from the `master` branch

Add a new function `repeat()` and use it

```
#!/usr/bin/env python2

def greetings():
    print "Hello RESIF people!"

def repeat(x, callback):
    for _ in range(x):
        callback()


if __name__ == "__main__":
    repeat(3, greetings)
```

As usual, commit our work

```
fabien@izanagi ~/my_awesome_project » git commit -a -m 'Add repeat() function'
[master 449e0a0] Add repeat() function
 1 file changed, 5 insertions(+), 1 deletion(-)
```

Two versions of our code

```
      ②
    * 449e0a0 - Add repeat() function                          (HEAD -> master)
    |                                                            ①
    |       ②
    |  * 34350f2 - Add Python3 support                                (python3)
    | /
    |/
    |
    |
    * b6cb2cc - What an improvement!
    |
    |
    |
    * 64188b7 - My first commit!
```

① Our last commit became the new HEAD of the repository

② Now we have two versions of our code, one in the `master` and a second one in the `python3` branch

Branches are useful when some modifications are difficult to implement (tricky bugs, big features), it's a way to store the work in progress without breaking the rest of the code

2.6. Merging

It's time to bring back Python3 support to our main branch `master`. This is done using the `git merge` command.

First we can check the differences between the two branches

```
fabien@izanagi Sync/my_awesome_project (master) » git diff master python3
diff --git a/app.py b/app.py
index f2aa257..436cd75 100755
--- a/app.py
+++ b/app.py
@@ -1,12 +1,8 @@
-#!/usr/bin/env python2
+#!/usr/bin/env python

 def greetings():
-    print "Hello RESIF people!"
-
-def repeat(x, callback):
-    for _ in range(x):
-        callback()
+    print("Hello RESIF people!")


 if __name__ == "__main__":
-    repeat(3, greetings)
+    greetings()
```

Now, let's try to merge python3 and master

```
fabien@izanagi ~/my_awesome_project » git checkout master ①
Already on 'master'
fabien@izanagi ~/my_awesome_project » git merge python3
Auto-merging app.py
Merge made by the 'recursive' strategy.
 app.py | 4 ++--
 1 file changed, 2 insertions(+), 2 deletions(-)
```

① Be sure to checkout the branch which will receive the changes

Did it work ?

```
fabien@izanagi ~/my_awesome_project (master) » python app.py
Hello RESIF people!
Hello RESIF people!
Hello RESIF people!
```

What happened

```
      ①
*   6ebe8a3 - Merge branch 'python3'                          (HEAD -> master)
|\
| |
| |
| * 34350f2 - Add Python3 support                                   (python3)
| |
| |
| |
* | 449e0a0 - Add repeat() function
|/
|
|
* b6cb2cc - What an improvement!
|
|
|
* 64188b7 - My first commit!
```

① The merge created a new commit on `master`

2.7. Restore a file

You've started to work on your code

```
fabien@namazu ~/my_awesome_project (master *) » git diff
diff --git a/app.py b/app.py
index aac437d..741b2f6 100755
--- a/app.py
+++ b/app.py
@@ -7,6 +7,9 @@ def repeat(x, callback):
     for _ in range(x):
         callback()

+def an_useless_function(message):
+    print(message)
+

 if __name__ == "__main__":
     repeat(3, greetings)
```

Maybe not enough coffee this morning, you realize your function pretty useless, let's restore the last version committed in our repository

```
fabien@namazu ~/my_awesome_project (master *) » git checkout app.py ①
fabien@namazu ~/my_awesome_project (master) » git diff ②
```

① We ask to git to checkout the last version of app.py

② We verify that there is no more modifications

You can restore any version of your code

List your commits

```
fabien@namazu ~/my_awesome_project (master) » git log --oneline
6ebe8a3 Merge branch 'python3'
449e0a0 Add repeat() function
34350f2 Add Python3 support
b6cb2cc What an improvement!
64188b7 My first commit!
```

Want to go back to the very first version ?

```
fabien@namazu ~/my_awesome_project (master) » git checkout 64188b7
Note: checking out '64188b7'.

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 64188b7... My first commit!
```

Your old code is back

```
fabien@namazu ~/my_awesome_project (HEAD (no branch)) » cat app.py
#!/usr/bin/env python2

def greetings():
    print "Hello world!"


if __name__ == "__main__":
    greetings()
```

You can switch back to the last version

```
fabien@namazu ~/my_awesome_project (HEAD (no branch)) » git checkout master
Previous HEAD position was 64188b7... My first commit!
Switched to branch 'master'
fabien@namazu ~/my_awesome_project (master) » cat app.py
#!/usr/bin/env python

def greetings():
    print("Hello RESIF people!")

def repeat(x, callback):
    for _ in range(x):
        callback()


if __name__ == "__main__":
    repeat(3, greetings)
```

Go back to our second commit !

```
fabien@namazu ~/my_awesome_project (master) » git checkout b6cb2cc
Note: checking out 'b6cb2cc'.

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 b6cb2cc... What an improvement!
fabien@namazu ~/my_awesome_project (HEAD (no branch)) » cat app.py
#!/usr/bin/env python2

def greetings():
    print "Hello RESIF people!"


if __name__ == "__main__":
    greetings()
```

Then return to our last version ...

```
fabien@namazu ~/my_awesome_project (HEAD (no branch)) » git checkout master
Previous HEAD position was b6cb2cc... What an improvement!
Switched to branch 'master'
```

As the `cd` command, you can use `-` to go back to the previous checkout : `git checkout -`

You can also restore specific files from an old changeset

```
fabien@namazu ~/my_awesome_project (master +) » git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   app.py

fabien@namazu ~/my_awesome_project (master +) » git reset HEAD app.py
Unstaged changes after reset:
M   app.py
fabien@namazu ~/my_awesome_project (master *) » git checkout app.py
```

As you don't checkout the whole repository, choosen files will be directly staged for the next commit (if they are different from the last knew version). That's why we need these additional commands to revert back the checkout.

2.8. Partial commit

There is a lack of documentation on your project, let's start a new branch to start documentation

```
fabien@namazu ~/my_awesome_project (master) » git checkout -b documentation
Switched to a new branch 'documentation'
```

And update `app.py` *to add some docstrings*

```
#!/usr/bin/env python

def greetings():
    """Salute RESIF people."""
    print("Hello RESIF people!")

def repeat(x, callback):
    """Call x times callback."""
    for _ in range(x):
        callback()


if __name__ == "__main__":
    repeat(3, greetings)
```

As we comment two functions, we could split our work into two commits

```
fabien@namazu ~/my_awesome_project (documentation *) » git add --patch app.py
diff --git a/app.py b/app.py
index aac437d..6a1e14a 100755
--- a/app.py
+++ b/app.py
@@ -1,9 +1,11 @@
 #!/usr/bin/env python

 def greetings():
+    """Salute RESIF people."""
     print("Hello RESIF people!")

 def repeat(x, callback):
+    """Call x times callback."""
     for _ in range(x):
         callback()

Stage this hunk [y,n,q,a,d,/,s,e,?]? s
Split into 2 hunks.
@@ -1,6 +1,7 @@
 #!/usr/bin/env python

 def greetings():
+    """Salute RESIF people."""
     print("Hello RESIF people!")

 def repeat(x, callback):
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? y
@@ -4,6 +5,7 @@
     print("Hello RESIF people!")

 def repeat(x, callback):
+    """Call x times callback."""
     for _ in range(x):
         callback()

Stage this hunk [y,n,q,a,d,/,K,g,e,?]? q
```

Verify the status of your repository

```
fabien@namazu ~/my_awesome_project (documentation) » git status
On branch documentation
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   app.py ①

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   app.py ②
```

① A part of our file is staged for the next commit…

② …but not all the file as we wanted.

Commit our changes

```
fabien@namazu ~/my_awesome_project (documentation) » git diff --staged
diff --git a/app.py b/app.py
index aac437d..ada6b00 100755
--- a/app.py
+++ b/app.py
@@ -1,6 +1,7 @@
 #!/usr/bin/env python

 def greetings():
+    """Salute RESIF people."""
     print("Hello RESIF people!")

 def repeat(x, callback):
fabien@namazu ~/my_awesome_project (documentation) » git commit -m 'Add docstring to
greetings()'
[documentation 942fddd] Add docstring to greetings()
 1 file changed, 1 insertion(+)
fabien@namazu ~/my_awesome_project (documentation *) » git diff
diff --git a/app.py b/app.py
index ada6b00..6a1e14a 100755
--- a/app.py
+++ b/app.py
@@ -5,6 +5,7 @@ def greetings():
     print("Hello RESIF people!")

 def repeat(x, callback):
+    """Call x times callback."""
     for _ in range(x):
         callback()

fabien@namazu ~/my_awesome_project (documentation *) » git commit -a -m 'Add docstring
to repeat()'
[documentation bee19c3] Add docstring to repeat()
 1 file changed, 1 insertion(+)
```

2.9. Squashing commits

Finally we decide that to have a documentation branch is non-sense and two commits is overkill.

First have a look to our repository

```
    * bee19c3 - Add docstring to repeat()                    (HEAD -> documentation)
    |
    |
    |
    * 942fddd - Add docstring to greetings()
    |
    |
    |
    *   6ebe8a3 - Merge branch 'python3'                                (master)
    |\
    | |
    | |
    | * 34350f2 - Add Python3 support                                 (python3)
    | |
    | |
    | |
    * | 449e0a0 - Add repeat() function
    |/
    |
    |
    * b6cb2cc - What an improvement!
    |
    |
    |
    * 64188b7 - My first commit!
```

Let's go back to our master branch

```
fabien@namazu ~/my_awesome_project (documentation) » git checkout master
Switched to branch 'master'
```

Merge using --squash option

```
fabien@izanagi ~/my_awesome_project (master) » git merge --squash documentation
Updating 6ebe8a3..bee19c3
Fast-forward
Squash commit -- not updating HEAD
 app.py | 2 ++
 1 file changed, 2 insertions(+)
```

What we get ?

```
fabien@izanagi ~/my_awesome_project (master +) » git diff --staged
diff --git a/app.py b/app.py
index aac437d..6a1e14a 100755
--- a/app.py
+++ b/app.py
@@ -1,9 +1,11 @@
 #!/usr/bin/env python

 def greetings():
+    """Salute RESIF people.""" ①
     print("Hello RESIF people!")

 def repeat(x, callback):
+    """Call x times callback.""" ①
     for _ in range(x):
         callback()
```

① All the modifications from our documentation branch are staged in master branch

Commit the changes

```
fabien@izanagi ~/my_awesome_project (master +) » git commit -a -m 'Add docstrings'
[master 4035a35] Add docstrings
 1 file changed, 2 insertions(+)
```

Did we really merge `documentation` branch ?

```
    ②
* 4035a35 - Add docstrings                                    (HEAD -> master)
|
|
|     ①
| * bee19c3 - Add docstring to repeat()                        (documentation)
| |
| |
| |
| * 942fddd - Add docstring to greetings()
|/
|
|
*   6ebe8a3 - Merge branch 'python3'
|\
| |
| |
| * 34350f2 - Add Python3 support                                    (python3)
| |
| |
| |
* | 449e0a0 - Add repeat() function
|/
|
|
* b6cb2cc - What an improvement!
|
|
|
* 64188b7 - My first commit!
```

① `merge --squash` only retrieved changes from `documentation` branch but didn't create a merge relationship. We could delete `documentation` branch using the following command : `git branch -D documentation`

② Our commit containing all the changes from `documentation` branch

2.10. Overview of git operations