Introduction to Git

Anthony Scemama <scemama@irsamc.ups-tlse.fr>

Labratoire de Chimie et Physique Quantiques
IRSAMC (Toulouse)

Version control systems

- Git is a Version Control System (VCS)
- Other VCS : Subversion, Mercurial, CVS, Bazaar, Darcs, ...
- Saves all the history of your your program (even years ago) as snapshots

One purpose

```
$ tar -zcf $(date +"%Y_%m_%d")_my_code.tar.gz \
  --exclude="*.o" my_code
2015_01_21_my_code.tar.gz
```

But also:

- Merge modifications of multiple users
- Handle different forks of the source files
- Make a centralized repository
- Small storage space

Git

All the history is stored *locally* on the computer of each user, even if a central repository is set up.

Benefits:

- Network is not needed for common tasks (you can work in the plane)
- Every user has a copy of the whole repository : distributed backups

To minimize storage, only differences (`diff`) are stored in a compressed database. The storage is very low (if only text files are tracked).

Everything stored in a git repository is always accessible.

Three states

Modified

files have been modified from the last saved commit (snapshot)

Staged

Marked to be saved in the next commit

Committed

Saved in the database

The modification of the state of the files is done using the *git* shell command.

Each commit is identified by a unique SHA1 key, for example

```
6dba5ebe21356a96fc8d0048758fda81215eec5f
```

Initial Git configuration

Installation:

```
$ sudo apt-get install git    # Debian, Ubuntu, ...
$ sudo yum install git        # CentOS, RedHat, Fedora, ...
```

Git tracks the modifications made by multiple users on the same project.

The first thing you have to do is to tell Git your name and email address:

```
$ git config --global user.name "Anthony Scemama"
$ git config --global user.email "scemam@irsamc.ups-tlse.fr"
```

The effect of this command is to write the `$HOME/.gitconfig` file:

```
[user]
      name = Anthony Scemama
      email = scemama@irsamc.ups-tlse.fr
```

Git commands

To get help, type `git help`:

```
$ git help
usage: git [--version] [--exec-path[=<path>]] [--html-path]
           [--man-path] [--info-path]
           [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [-c name=value] [--help]
           <command> [<args>]

The most commonly used git commands are:
   add        Add file contents to the index
   bisect     Find by binary search the change that introduced a bug
   branch     List, create, or delete branches
   checkout   Checkout a branch or paths to the working tree
   clone      Clone a repository into a new directory
```

```
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   fetch      Download objects and refs from another repository
   grep       Print lines matching a pattern
   init       Create an empty git repository or reinitialize an existing one
   log        Show commit logs
   merge      Join two or more development histories together
   mv         Move or rename a file, a directory, or a symlink
   pull       Fetch from and merge with another repository or a local branch
   push       Update remote refs along with associated objects
   rebase     Forward-port local commits to the updated upstream head
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index
   show       Show various types of objects
   status     Show the working tree status
   tag        Create, list, delete or verify a tag object signed with GPG

See 'git help <command>' for more information on a specific command.
```

To get help on a particular command, type `git help <command>`.

Initialization of a Project

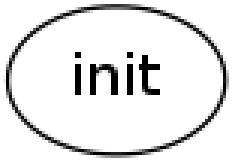
Use `git init` to initialize a new repository.

```
$ mkdir NewProject

$ cd NewProject

$ git init
Initialized empty Git repository in /home/scemama/NewProject/.git/

$ ls

$ ls -a
.  ..  .git

$ ls .git/
branches  config  description  HEAD  hooks  info  objects  refs
```

A `.git` directory is created, containing all the database.

To see the status of the git repository, use `git status`:

```
$ git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```



Adding files to the repository

Now, create a file `test.f90`:

```
program test
  implicit none
  print *, 'Hello world'
end
```

Tell git to track the `test.f90` file:

```
$ git add test.f90
```

Compile the program, and check the status of your repository using `git status`:

```
$ gfortran test.f90

$ git status
# On branch master
```

```
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   test.f90
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       a.out
```

The `test.f90` is now **staged**.

Telling git to ignore some files

The `a.out` file will never be tracked by git. You can tell it to git by adding `a.out`, as well as all the `*.o` files to the `.gitignore` file.

```
$ cat << EOF > .gitignore
a.out
*.o
EOF

$ git add .gitignore
```

```
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   .gitignore
#       new file:   test.f90
#
```

Making a commit

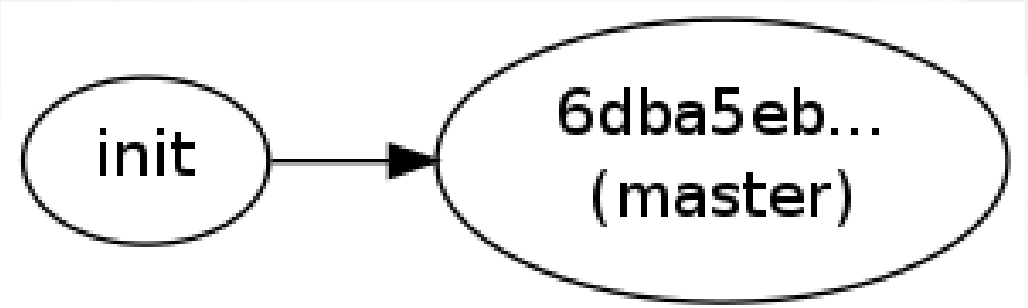
When you create a commit, you *need* to give a documentation for the commit.

```
$ git commit -m "Hello world program"
[master (root-commit) 6dba5eb] Hello world program
 2 files changed, 7 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 test.f90

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

To see the git history, run `git log`:

```
$ git log
commit 6dba5ebe21356a96fc8d0048758fda81215eec5f
Author: Anthony Scemama <scemama@irsamc.ups-tlse.fr>
```

```
Date:   Thu Jan 22 13:20:25 2015 +0100

    Hello world program
```



Displaying differences

Now you can modify the `test.f90` file:

```
program test
  implicit none
  print *, 'Bonjour'
end
```

Git sees that the file has been modified:

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   test.f90
#
no changes added to commit (use "git add" and/or "git commit -a")
```

The `test.f90` is now in the **modified** state.

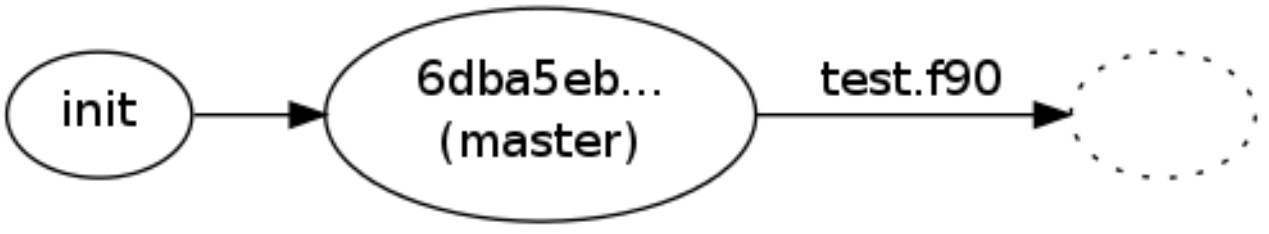
To show what has been modified, run `git diff`:

```
$ git diff
diff --git a/test.f90 b/test.f90
index 7231a91..1f01d3b 100644
--- a/test.f90
+++ b/test.f90
@@ -1,4 +1,4 @@
 program test
  implicit none
-  print *, 'Hello world'
+  print *, 'Bonjour'
 end
```

Now, put the `test.f90` file in the **staged** state by running:

```
$ git add test.f90

$ git status
```

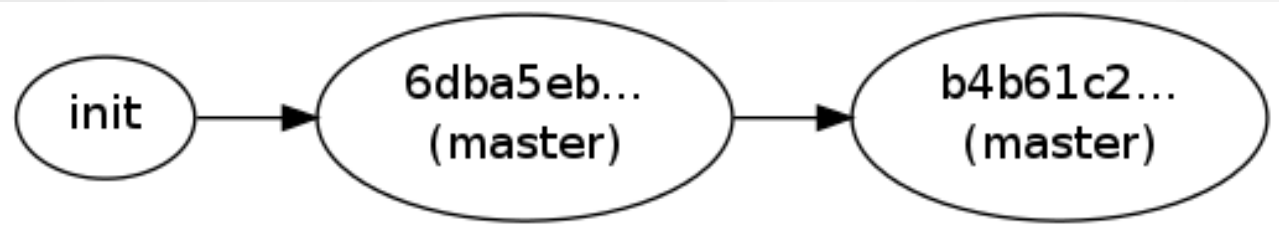
```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#     modified:   test.f90
#
```



Now, create the new commit

```
$ git commit -m "Switched to french"
[master b4b61c2] Switched to french
 1 file changed, 1 insertion(+), 1 deletion(-)

$ git log
```

```
commit b4b61c21c573b24345ace588df98287f92e819b6
Author: Anthony Scemama <scemama@irsamc.ups-tlse.fr>
Date:   Sat Jan 24 11:02:50 2015 +0100

    Switched to french

commit 6dba5ebe21356a96fc8d0048758fda81215eec5f
Author: Anthony Scemama <scemama@irsamc.ups-tlse.fr>
Date:   Thu Jan 22 13:20:25 2015 +0100

    Hello world program


$ git log --oneline
b4b61c2 Switched to french
6dba5eb Hello world program
```



To inspect a given commit, give the beginning of its SHA1 key to `git show`:

```
$ git show b4b61c2
commit b4b61c21c573b24345ace588df98287f92e819b6
Author: Anthony Scemama <scemama@irsamc.ups-tlse.fr>
Date:   Sat Jan 24 11:02:50 2015 +0100

    Switched to french

diff --git a/test.f90 b/test.f90
index 7231a91..1f01d3b 100644
--- a/test.f90
+++ b/test.f90
@@ -1,4 +1,4 @@
```

```
program test
  implicit none
-  print *, 'Hello world'
+  print *, 'Bonjour'
end
```

Creating a branch

Suppose you want to make a fork of the program starting from the "Hello world" version. You can create a new branch called *development* starting on commit `6dba5eb...`:

```
$ git branch
* master

$ git checkout -b development 6dba5eb
Switched to a new branch 'development'

$ git branch
* development
  master

$ git log
commit 6dba5ebe21356a96fc8d0048758fda81215eec5f
```

```
Author: Anthony Scemama <scemama@irsamc.ups-tlse.fr>
Date:   Thu Jan 22 13:20:25 2015 +0100

    Hello world program
```

Now you can add a Makefile for example:

```
a.out: test.f90
	gfortran test.f90
```

And commit your changes:

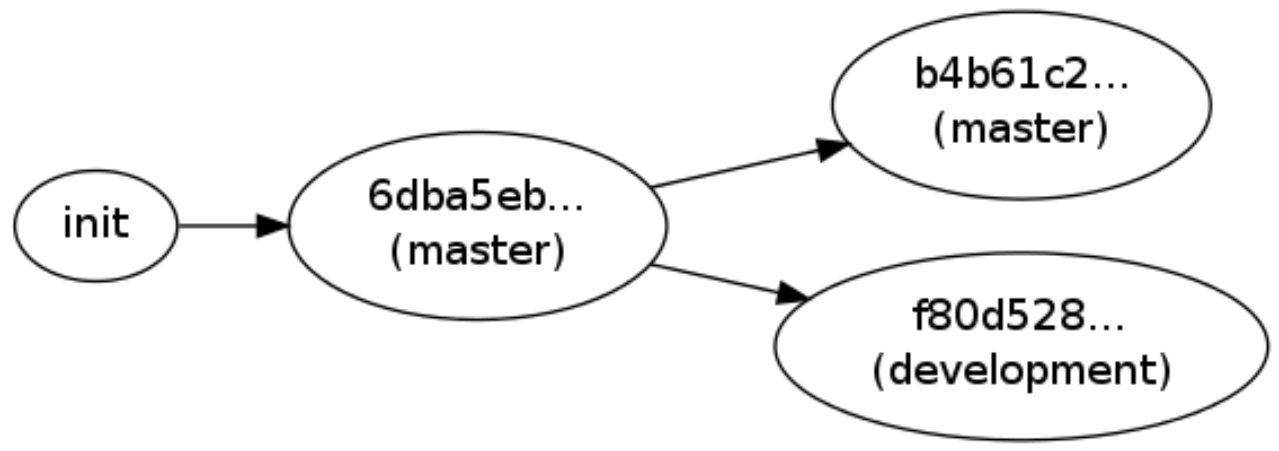
```
$ git add Makefile

$ git commit -m "Added Makefile"
[development f80d528] Added Makefile
1 file changed, 2 insertions(+)
create mode 100644 Makefile
```

```
$ git log --oneline
f80d528 Added Makefile
6dba5eb Hello world program
```

And you can switch very easily from one branch to another using `git checkout`:

```
$ ls
a.out  Makefile  test.f90

$ git checkout master
Switched to branch 'master'

$ git branch
  development
* master

$ ls
a.out  test.f90
```

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

$ ls
a.out  Makefile  test.f90
```



Merging branches

Let us now make the two branches evolve independently.

Work in the master branch

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

Create a `README.md` file:

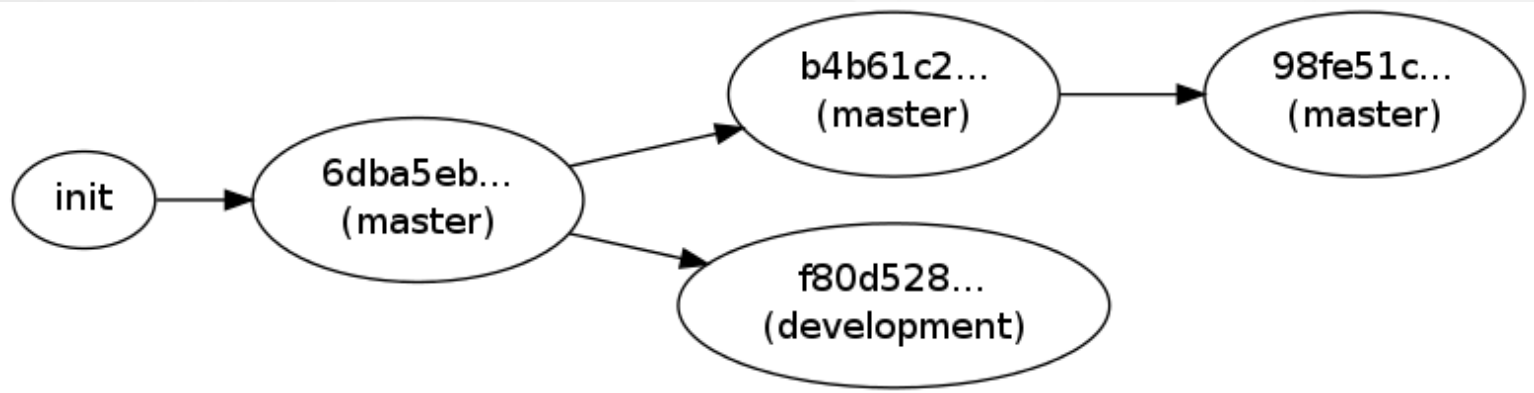
```
Bonjour
=======

Ce programme imprime bonjour a l'ecran.
```

and add a comment to the test.f90 file:

```
program test
  implicit none
  ! Ce programme imprime bonjour
  print *, 'Bonjour'
end
```

Commit everything in the master branch:

```
$ git add README.md
$ git commit -a -m "Added README file"
[master 98fe51c] Added README file
2 files changed, 7 insertions(+)
create mode 100644 README.md

$ git log --oneline
98fe51c Added README file
b4b61c2 Switched to french
6dba5eb Hello world program
```



Work in the development branch

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

Add a subroutine call in the main program:

```
program test
  implicit none
  print *, 'Hello world'
  call my_subroutine
end
```

And create a new file `sub.f90` containing the subroutine:

```
subroutine my_subroutine
  print *,  'called my_subroutine'
end
```

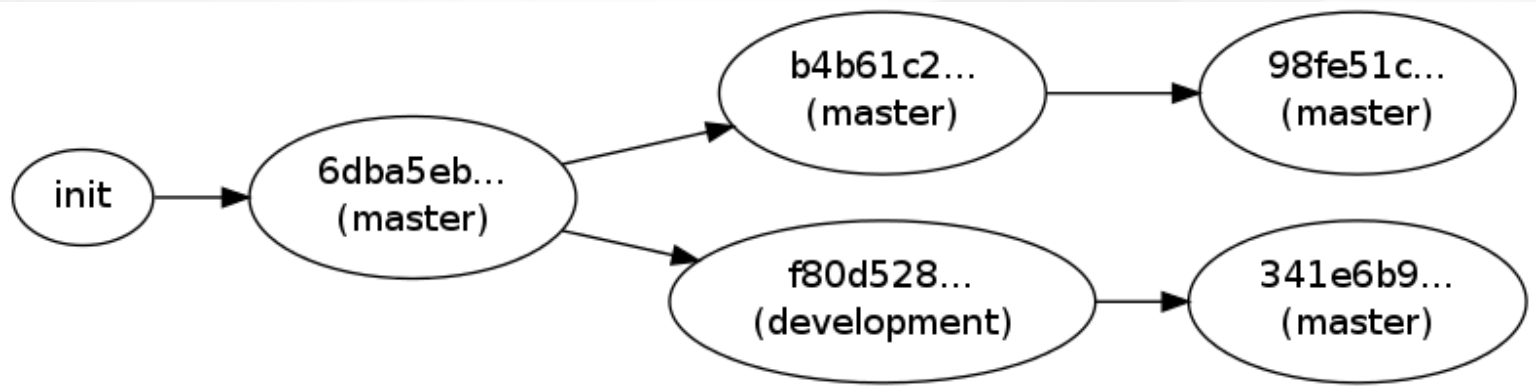
Modify the Makefile accordingly:

```
a.out: test.f90 sub.f90
	gfortran test.f90 sub.f90
```

And commit everything:

```
$ git add sub.f90 Makefile test.f90

$ git commit -m "Introduced subroutine"
[development 341e6b9] Introduced subroutine
3 files changed, 6 insertions(+), 2 deletions(-)
create mode 100644 sub.f90
```

```
$ git log --oneline
341e6b9 Introduced subroutine
f80d528 Added Makefile
6dba5eb Hello world program
```



Merging

We now want to merge the development *on* the master branch.

First, checkout the master branch:

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

Then do the merge:

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

```
$ git merge development
Auto-merging test.f90
CONFLICT (content): Merge conflict in test.f90
Automatic merge failed; fix conflicts and then commit the result.
```

It failed: git can't automatically merge the files because there are conflicts in `test.f90`:

```
program test
  implicit none
<<<<<<< HEAD
  ! Ce programme imprime bonjour
```

```
  print *, 'Bonjour'
=======
  print *, 'Hello world'
  call my_subroutine
>>>>>>> development
end
```

Resolve the conflicts by hand:

```
program test
  implicit none
  ! Ce programme imprime bonjour
  print *, 'Bonjour'
  call my_subroutine
end
```

```
$ git status
# On branch master
# Changes to be committed:
```

```
#
#       new file:   Makefile
#       new file:   sub.f90
#
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate to mark resolution)
#
#       both modified:      test.f90
#
```

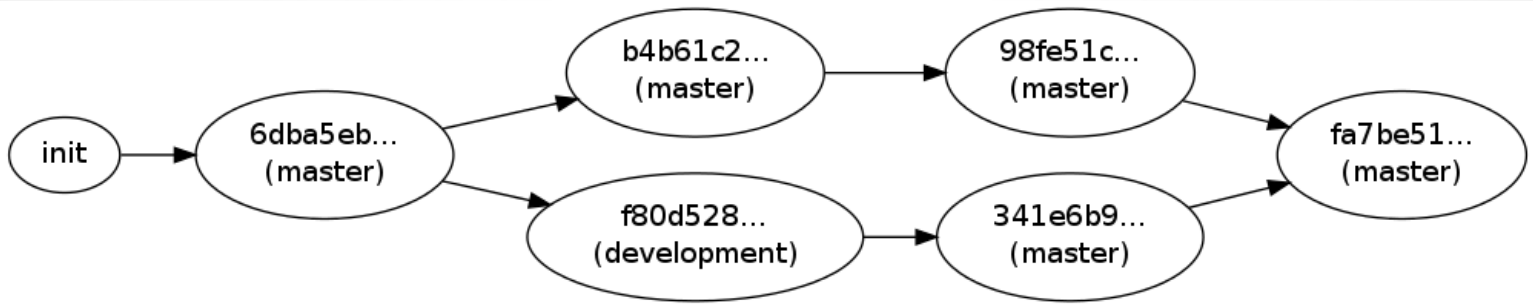
Commit your changes:

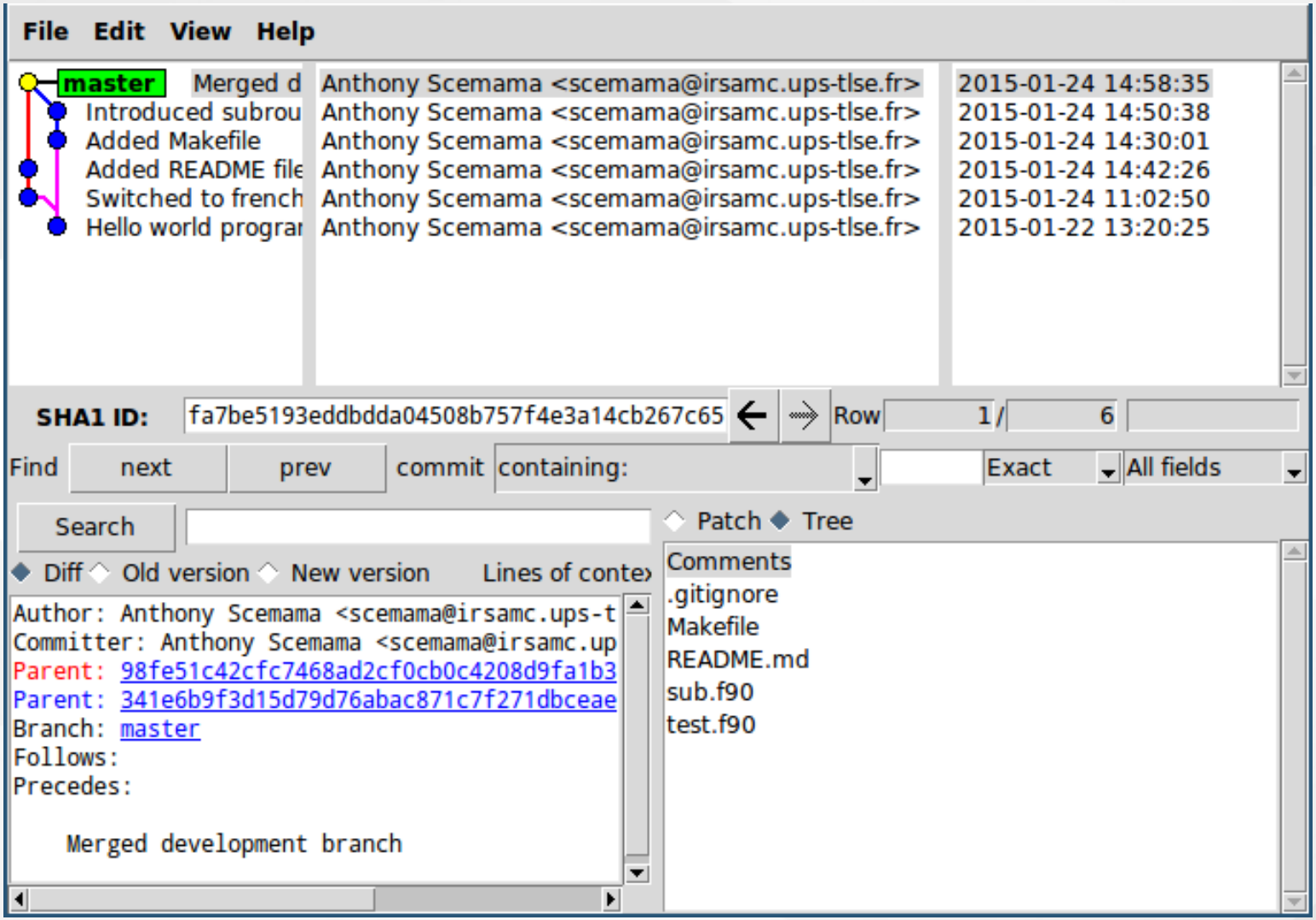
```
$ git commit -a -m "Merged development branch"
[master fa7be51] Merged development branch

$ ls
a.out  Makefile  README.md  sub.f90  test.f90

$ git log --oneline
fa7be51 Merged development branch
```

```
341e6b9 Introduced subroutine
98fe51c Added README file
f80d528 Added Makefile
b4b61c2 Switched to french
6dba5eb Hello world program
```

You can now delete the `development` branch:

```
$ git branch -d development
Deleted branch development (was 341e6b9).

$ git branch
* master
```




File Edit View Help
master Merged d
Introduced subrou
Added Makefile
Added README file
Switched to french
Hello world prograr
Anthony Scemama <scemama@irsamc.ups-tlse.fr>
2015-01-24 14:58:35
2015-01-24 14:50:38
2015-01-24 14:30:01
2015-01-24 14:42:26
2015-01-24 11:02:50
2015-01-22 13:20:25
SHA1 ID: fa7be5193eddbdda04508b757f4e3a14cb267c65
Row 1 / 6
Find next prev commit containing:
Exact
All fields
Search
Patch Tree
Diff Old version New version Lines of conte
Author: Anthony Scemama <scemama@irsamc.ups-t
Committer: Anthony Scemama <scemama@irsamc.up
Parent: 98fe51c42cfc7468ad2cf0cb0c4208d9fa1b3
Parent: 341e6b9f3d15d79d76abac871c7f271dbceae
Branch: master
Follows:
Precedes:
Merged development branch
Comments
.gitignore
Makefile
README.md
sub.f90
test.f90

Fetching files from a previous commit

If you want to go back *temporarily* to a given commit (`98fe51c`), you can do:

```
$ git checkout 98fe51c42cfc7468ad2cf0cb0c4208d9fa1b3a5a
Note: checking out '98fe51c42cfc7468ad2cf0cb0c4208d9fa1b3a5a'.

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 98fe51c... Added README file
```

A temporary branch has been created:

```
$ git branch
* (no branch)
```

```
  master
```

To go back to the real work, you can go to the master branch again:

```
$ git checkout master
Previous HEAD position was 98fe51c... Added README file
Switched to branch 'master'

$ git branch
* master
```

And the temporary branch has disappeared.

You can also fetch some files of a given commit. For example, let's check how `test.f90` has changed since `98fe51c42`:

```
$ git diff 98fe51c42cfc7468ad2cf0cb0c4208d9fa1b3a5a test.f90
diff --git a/test.f90 b/test.f90
index 2e3ebd6..10f2638 100644
--- a/test.f90
```

```
+++ b/test.f90
@@ -2,4 +2,5 @@ program test
  implicit none
  ! Ce programme imprime bonjour
  print *, 'Bonjour'
+  call my_subroutine
end
```

Now, let's fetch the old version of the `test.f90` file:

```
$ git checkout 98fe51c42cfc7468ad2cf0cb0c4208d9fa1b3a5a test.f90

$ git diff 98fe51c42cfc7468ad2cf0cb0c4208d9fa1b3a5a test.f90

$ git diff

$ git status
# On branch master
# Changes to be committed:
```

```
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   test.f90
#
```

Warning: `test.f90` is automatically in the **Staged** state.

Undoing changes

Finally, we changed our mind and want to undo our staged changes.

If we do the following:

```
$ git reset
Unstaged changes after reset:
M       test.f90
```

everything is unstaged and the files are in the **Modified** state.

```
$ git diff
diff --git a/test.f90 b/test.f90
index 10f2638..2e3ebd6 100644
--- a/test.f90
+++ b/test.f90
@@ -2,5 +2,4 @@ program test
  implicit none
  ! Ce programme imprime bonjour
```

```
  print *, 'Bonjour'
-  call my_subroutine
end
```

If we had done this:

```
$ git reset --hard
HEAD is now at fa7be51 Merged development branch

$ git diff
```

everything would also have been unstaged, but all your changes would have been lost.

Cloning a repository

Suppose you have a repository on your desktop computer, and you want to have a copy of it on your laptop. You can use ssh to make a clone of your directory:

```
scemama@laptop $ git clone scemama@desktop:/home/scemama/NewProject
Cloning into 'NewProject'...
remote: Counting objects: 22, done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 22 (delta 4), reused 0 (delta 0)
Receiving objects: 100% (22/22), done.
Resolving deltas: 100% (4/4), done.

scemama@laptop $ cd NewProject

scemama@laptop $ git log --oneline
fa7be51 Merged development branch
341e6b9 Introduced subroutine
```

```
98fe51c Added README file
f80d528 Added Makefile
b4b61c2 Switched to french
6dba5eb Hello world program
```

And now you can do all your work locally on your laptop. Let's modify the `README.md` file:

```
Bonjour
=======

Ce programme imprime bonjour a l'ecran.
Bla bla bla....
....
...bla bla bla.
```

```
scemama@laptop $ git commit -a -m "Modification on laptop"
[master de98d07] Modification on laptop
 1 file changed, 3 insertions(+)
```

Now, on your desktop, you can `pull` the changes you have made on the laptop:

```
scemama@desktop $ git pull scemama@laptop:NewProject
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From localhost:/home/scemama/Seafile/TEX/Cours/EDSDM/Git/NewProject2
* branch            HEAD       -> FETCH_HEAD
Updating fa7be51..cb81044
Fast-forward
README.md |    3 +++
1 file changed, 3 insertions(+)
```

Creating a central repository

1. Create a *bare* repository :

```
$ git clone --bare --shared NewProject
Cloning into bare repository 'NewProject.git'...
done.
```

2. Copy the bare repository to some location:

```
$ scp -r NewProject.git scemama@desktop:GitRepositories/
$ rm -rf NewProject.git
```

3. Now tell git that this repo is the main repository:

```
$ cd NewProject
$ git remote add -t master origin scemama@desktop:GitRepositories/NewProject
$ git remote show origin
* remote origin
```

```
  Fetch URL: scemama@desktop:GitRepositories/NewProject
  Push  URL: scemama@desktop:GitRepositories/NewProject
  HEAD branch: master
  Remote branch:
    master tracked
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

4. Try to `pull` from this repository

```
$ git pull
Already up-to-date.
```

Working with a remote repository

1. Alice clones the repository

```
alice@laptop $ git clone alice@desktop:GitRepositories/NewProject
```

2. Alice starts working on the project

```
alice@laptop $ git commit -a -m "My first commit"
alice@laptop $ git show
commit ef2667ed1392fbe9518d671ff6ee781d843c8249
Author: Anthony Scemama <scemama@irsamc.ups-tlse.fr>
Date:   Mon Jan 26 15:16:17 2015 +0100

    My first commit

diff --git a/sub.f90 b/sub.f90
index f60ef24..3f18f42 100644
```

```
--- a/sub.f90
+++ b/sub.f90
@@ -1,3 +1,3 @@
subroutine my_subroutine
- print *,  'called my_subroutine'
+ print *,  '->  called my_subroutine'
end
```

3. Bob clones the repository

```
bob@laptop $ git clone bob@desktop:GitRepositories/NewProject
```

4. Bob starts working on the project

```
bob@laptop $ git commit -m "Commit from Bob"
bob@laptop $ git show
commit c0f6087f5e2120632745b3ee3d1297a17de0b315
Author: Bob <bob@ups-tlse.fr>
Date:   Mon Jan 26 15:19:56 2015 +0100
```

```
    Commit from Bob

diff --git a/test.f90 b/test.f90
index 10f2638..a166306 100644
--- a/test.f90
+++ b/test.f90
@@ -1,6 +1,9 @@
program test
  implicit none
  ! Ce programme imprime bonjour
+  integer :: i
  print *, 'Bonjour'
-  call my_subroutine
+  do i=1,10
+    call my_subroutine
+  enddo
end
```

5. Bob has finished implementing his feature. He pushes his modifications on the main repository

```
bob@laptop $ git push
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 376 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To localhost:GitRepositories/NewProject
  fa7be51..c0f6087  master -> master
```

6. Now, Alice has finished and wants to push her changes:

```
alice@laptop $ git push
To desktop:GitRepositories/NewProject
 ! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to 'desktop:GitRepositories/NewProject'
To prevent you from losing history, non-fast-forward updates were rejected
Merge the remote changes (e.g. 'git pull') before pushing again.  See the
'Note about fast-forwards' section of 'git push --help' for details.
```

The push is rejected because the remote repository has changed since her last update. She needs to merge her copy with the main repository before pushing:

```
alice@laptop $ git pull
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From desktop:GitRepositories/NewProject
  fa7be51..c0f6087  master     -> origin/master
Merge made by the 'recursive' strategy.
test.f90 |    5 ++++-
1 file changed, 4 insertions(+), 1 deletion(-)

alice@laptop $ git push
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 588 bytes, done.
```

```
Total 5 (delta 2), reused 0 (delta 0)
To desktop:GitRepositories/NewProject
  c0f6087..733f4fb  master -> master
```

Now, the repository contains the modifications of both Alice and Bob.

Finding bugs with a binary search

- You are discovering a bug that has been added in the past
- You have no idea when the bug was added
- You are sure that two years ago, the bug wasn't there
- You can use a binary search to find the bug in $\mathcal{O}(\log(N))$
- The machine is better than you for this job

```
$ ./my_program
 One plus one equals    1.0000000
```

On your current revision, start git bisect:

```
$ git bisect start
```

Tell git-bisect that the current revision is bad:

```
$ git bisect bad
```

Tell git-bisect that the revision two years ago was good

```
$ git bisect good 938ca4781610eeee9bf18abefc4c0da3229776bd
Bisecting: 161 revisions left to test after this (roughly 7 steps)
[35bd4cfd09390790c6c656813038f2b082243504] Random commit number 19172
```

```
$ make >/dev/null ; ./my_program
 One plus one equals    1.0000000
$ git bisect bad
Bisecting: 80 revisions left to test after this (roughly 6 steps)
[956cdd7991036540f250cdf075683502ef9a66b3] Random commit number 9520
$
$ make >/dev/null ; ./my_program
 One plus one equals    2.0000000
$ git bisect good
Bisecting: 40 revisions left to test after this (roughly 5 steps)
[17c741ffbf2f850aa319a46d02d8ec8471e30bf5] Random commit number 14280
...
$
```

```
$ make >/dev/null ; ./my_program
 One plus one equals    1.0000000
$
$ git bisect bad
61bc7a8a58684260ccf276dce4ed81207ffee409 is the first bad commit
commit 61bc7a8a58684260ccf276dce4ed81207ffee409
Author: Anthony Scemama <scemama@irsamc.ups-tlse.fr>
Date:   Tue Apr 9 12:07:00 2013 +0200

    Surprise...

:100644 100644 5334b6165ca4e33192717b47f48cc00eeebd76ac
 18f15065c486d68e456fafb6bd43c5c4b3c043ac M      main.f
```

- If compiling and testing takes 10 minutes, the bisection can take a few hours!
- Git bisect can be automated using a script
- If the exit code is zero, the test is good
- If the exit code is non-zero, the test is bad

```
#!/bin/bash
make >/dev/null
if [[ $? -ne 0 ]] ; then
        echo "WARNING : make failed"
        echo "Choosing randomly"
        exit $(( $RANDOM / 16384 ))
fi
result=$(./my_program | cut -b22-)
the_test=$(python -c "print $result == 2.")
if [[ $the_test = False ]] ; then
        exit 1
elif [[ $the_test = True ]] ; then
        exit 0
else
        echo "Python returned $the_test"
        exit $(( $RANDOM / 16384 ))
fi
```

```
$ git bisect start
$ git bisect bad
$ git bisect good 938ca4781610eeee9bf18abefc4c0da3229776bd
Bisecting: 161 revisions left to test after this (roughly 7 steps)
[35bd4cfd09390790c6c656813038f2b082243504] Random commit number 19172
$
$ time git bisect run ./bisect_run.sh
running ./bisect_run.sh
Bisecting: 80 revisions left to test after this (roughly 6 steps)
[956cdd7991036540f250cdf075683502ef9a66b3] Random commit number 9520
running ./bisect_run.sh
Bisecting: 40 revisions left to test after this (roughly 5 steps)
[17c741ffbf2f850aa319a46d02d8ec8471e30bf5] Random commit number 14280
...
```

```
running ./bisect_run.sh
61bc7a8a58684260ccf276dce4ed81207ffee409 is the first bad commit
commit 61bc7a8a58684260ccf276dce4ed81207ffee409
Author: Anthony Scemama <scemama@irsamc.ups-tlse.fr>
Date:   Tue Apr 9 12:07:00 2013 +0200
```

```
    Surprise...

:100644 100644 5334b6165ca4e33192717b47f48cc00eeebd76ac
18f15065c486d68e456fafb6bd43c5c4b3c043ac M      main.f
bisect run success

real    0m4.000s
user    0m2.910s
sys     0m0.210s
```

The bad commit was found in 4 seconds.

Note

You can use Git for anything :

- Programming
- Writing a thesis/papers
- Linux configuration files (`/etc/`)
- Calculations made sequentially (input and output files)
- Web pages and web site
- etc...