Practical Git

Alexis Praga

June 1, 2012

Why Git ?

Why do you want to use it ?

Killer features :

- decentralized
- light branches

Basic Git : Git in 30 seconds

```
$ git clone git@github.com:alexDarcy/gitdemo.git
Cloning into gitdemo...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Receiving objects: 100\% (3/3), 238 bytes, done.
```

```
$ cd gitdemo/
$ git add main.c
$ git commit -am "Main file."
[master 8a50278] Main file.
 1 files changed, 4 insertions(+), 0 deletions(-)
  create mode 100644 main.c
```

```
$ git push
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 336 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:alexDarcy/gitdemo.git
   2ca308c..8a50278  master -> master
```

Basics

Local commands

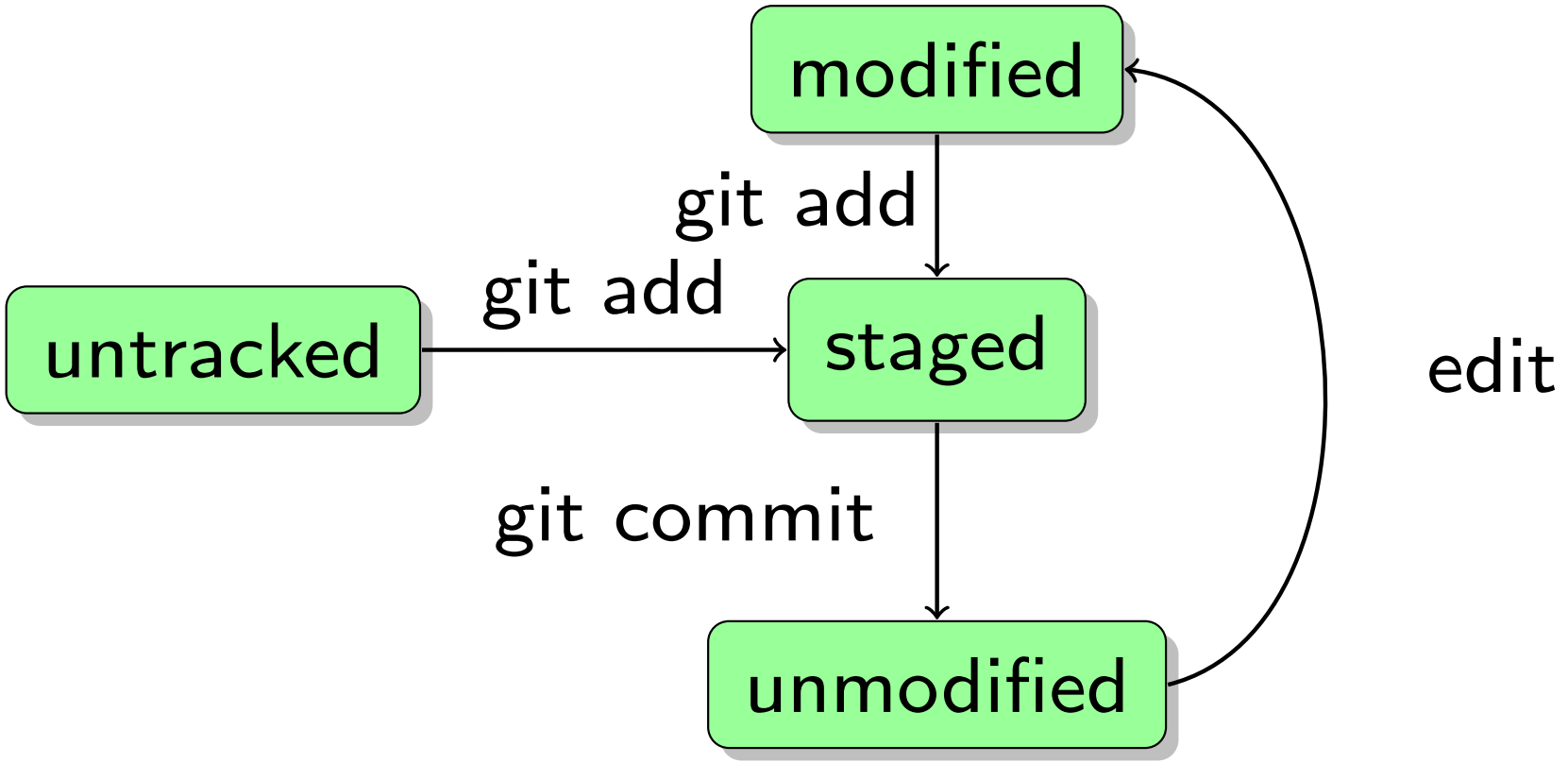
- `git add`
- `git commit`

Server communication

- `git push`
- `git pull`

Workflow

`git status` will tell you what to do.



Viewing History

- `git log` : commit message, author, date
- `git log -p` : shows diff

```
$ git log
commit 8a5027842b0ca71e462906384b91922537450858
Author: Alexis Praga <alexis.praga@free.fr>
Date:   Mon May 28 13:02:01 2012 +0200

    Main file.

commit 2ca308c8f79c718b66d6d90755561a53322e2717
Author: alexDarcy <alexis.praga@free.fr>
Date:   Mon May 28 03:59:02 2012 -0700

    Initial commit
```

- Graphical : `gitk`
- `git diff` : view exact changes

Undoing

- `git commit --amend` : add some file to a commit (staging area). Merge in Single commit
- `git reset HEAD myfile` : unstage
- `git checkout` : undo changes

Warning : hard to lose something, but it must be commited.

Tips

`git commit -am "Your commit message"` is a fast way to commit all your code.

Beware, it is better to commit specific changes.

Intermediate Git : Branching is easy

Branching

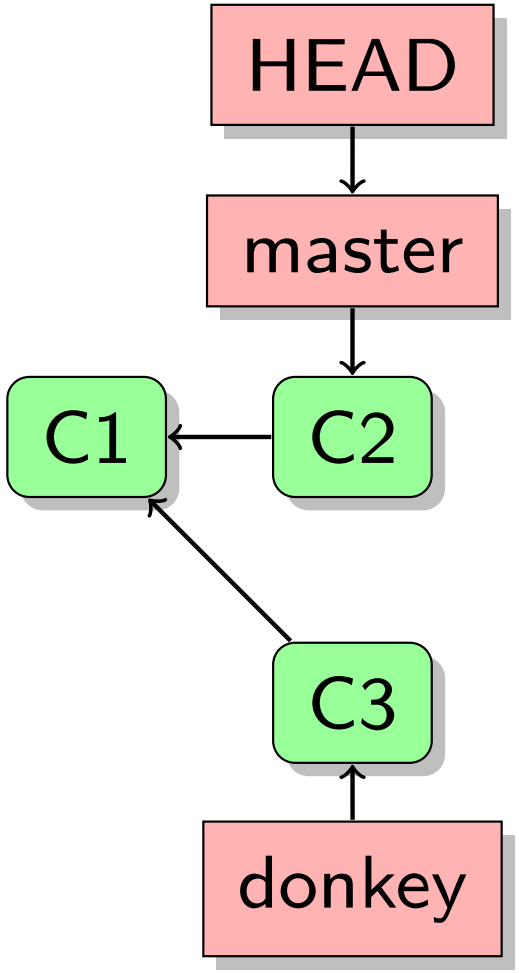
- `git branch monkey` : creates a branch
- `git checkout monkey` : switch to the branch
- `git checkout -b monkey` : both of the above

Merging

- `git checkout master`, `git merge monkey` : merge to master
- `git branch -d monkey` : delete the branch. (Don't worry !)

How does it work ?

A branch is just a pointer to a commit. HEAD is a pointer to the current branch.



Using branches

Managing

```
$ git branch -v
   master  69620a8 Merge remote branch 'origin'
 * testing 4a51090 Adding data
```

```
$ git log --graph --oneline --all
*   965c326 Merge branch 'testttttt'
|\
| * 7a8a59e New files
| * f91f1c0 Adding an useless file
| * ace4955 Merging the following fichier 3 (changed) with output1
* | d3966a6 output2
* | 2b4181a output1
* | 4ed5e08 fichier 3
|/
* 82b57e3 fichier 2
```

Using branches (2)

Workflow

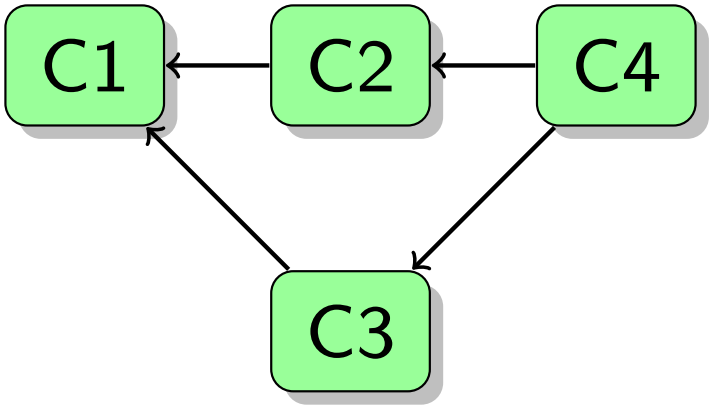
- Long-term branches. Ex (Debian) : `stable`,`testing`,`unstable`
- Short-term branches : for testing ideas locally.

Tracking branch

Ex : cloning creates a master branch which will track origin/master

- `git checkout --track origin/monkey` : will track monkey
- `git checkout -b donkey origin/monkey` : idem but with another name.
- `git push origin :monkey` : delete remote branch (! difficult to memorise)

Merging and conflicts




```
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Don't panic !

Simply edit the files and keep the sections you want.

```
<<<<<<< HEAD:index.html
j = cos(i*M_PI)*sin(i_MPI/3.)
=======
j = i_MPI/3.
>>>>>>> iss53:index.html
```

Then check everything has been merged : `git status`, and `git commit` (default message).

Tips

With git merge, it will automatically commit. Use the `--no-commit` option if you don't want to.

`git mergetool` for a visual merging tool.

Merging strategies

Tired of conflicts ? You can accept unilateraly one branche :

- `git merge -s ours`

`git merge -s theirs` does not exist in new versions. You can emulate its behaviour with :

```
git checkout -b devel origin/upstream
git merge -s ours toignore
git checkout toignore
git merge devel
```

The last two commands allows a fast-forward to the HEAD of `toignore`. However, it is cleaner to do

```
$ git fetch origin
$ git reset --hard origin
```

If you want to keep your work, just create a branch before the `reset`.

Remote branches

Warning : source of confusion.

What happens when I clone ?

Cloning creates a remote (origin) and a branch (master) : origin/master. push and pull with use this. **But** we will work on the local master branch.

Create your own remote

- `git remote -v` : show remote
- `git remote add [shortname] [url]`: add a remote
- `git push [remote] [branch]`: push to the remote.
- `git remote show` : show some informations about the remote
- `git remote rm`
- `git remote rename`

Remote branches (2)

Quizz : What if someone push before me ?

Remote branches (2)

Quizz : What if someone push before me ?

```
To git@github.com:alexDarcy/gitdemo.git
 ! [rejected]        master -> master (non-fast-forward)
 error: failed to push some refs to 'git@github.com:alexDarcy/
 gitdemo.git'
 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.
```

Remote branches (2)

Quizz : What if someone push before me ?

```
To git@github.com:alexDarcy/gitdemo.git
 ! [rejected]        master -> master (non-fast-forward)
 error: failed to push some refs to 'git@github.com:alexDarcy/
 gitdemo.git'
 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.
```

Walkthrough

- Quickfix : `git pull` and `git push`
- Longer solution :
  - `git fetch`
  - `git log HEAD..origin`
  - `git merge origin`
  - `git push`

Remote branches (3)

How to pull from a certain commit ?

```
$ git fetch
$ git log HEAD..origin
$ git merge cae685f
```

Only for commit newer than HEAD.

Advanced Git : Rebase

Rewrites history as if we did only successive commits.

- `git checkout donkey` and `git rebase master`
- `git checkout master` and `git merge donkey`

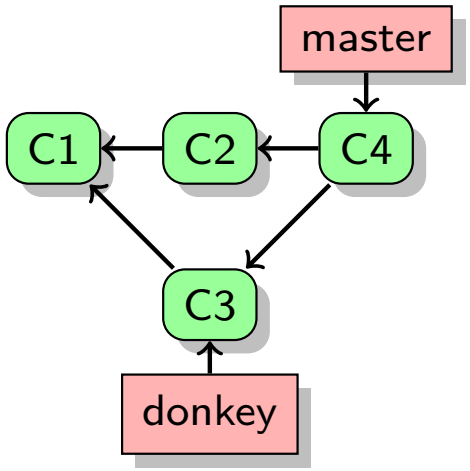


Figure: Merge

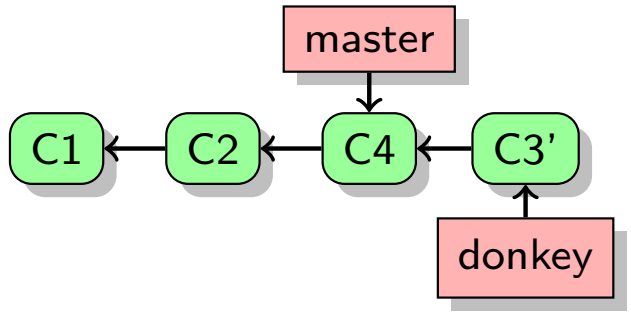


Figure: Rebase

Quizz : For what purpose ?

Advanced Git : Rebase

Rewrites history as if we did only successive commits.

- `git checkout donkey` and `git rebase master`
- `git checkout master` and `git merge donkey`

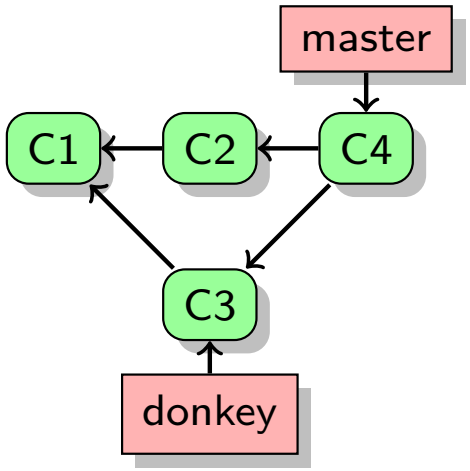


Figure: Merge

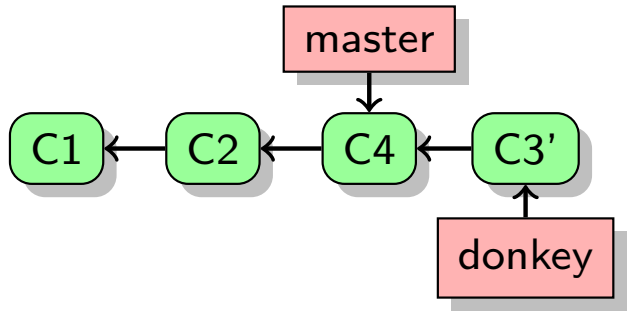


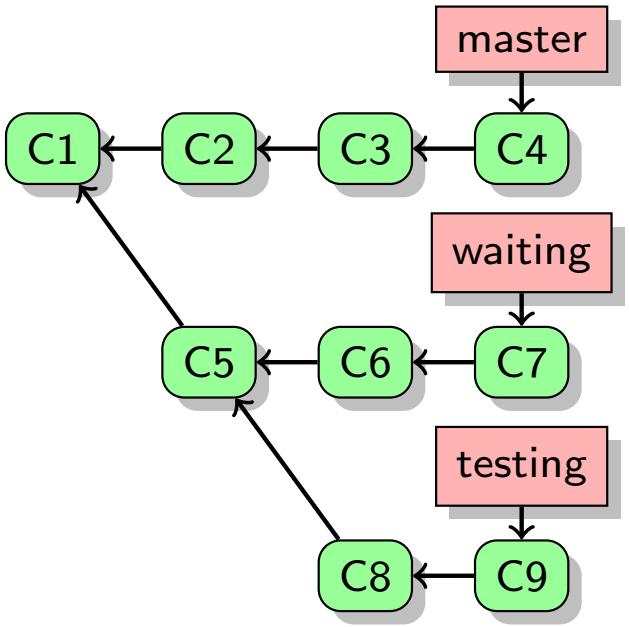
Figure: Rebase

Quizz : For what purpose ?

Test your branch integrates smoothly to master.

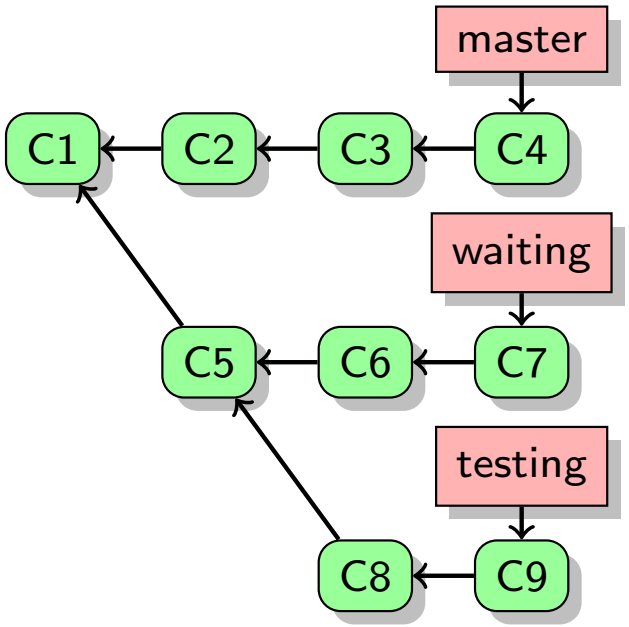
Rebase (2)

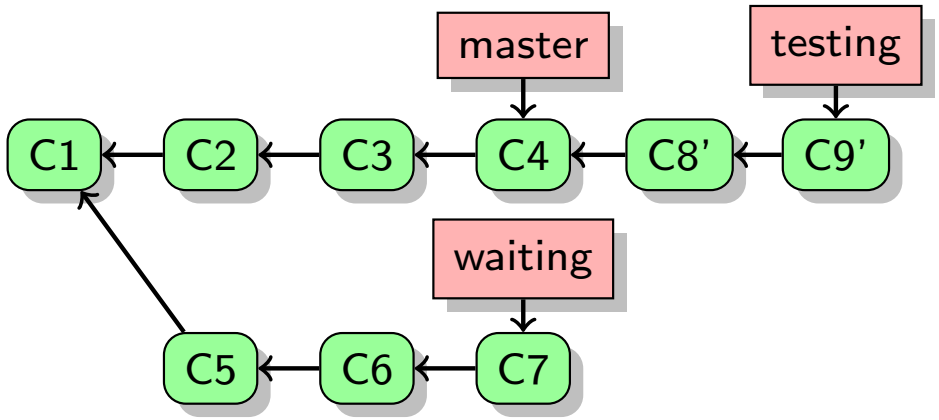
Quizz : What is the output of `git rebase --onto master waiting testing` ?



Rebase (2)

Quizz : What is the output of `git rebase --onto master waiting testing` ?





Rebasing (3)

```
* 5e3ef1b output2
* 29ee50d output1
| * 79f5ba4 data2
| * d625223 data 1
|/
* 4a51090 Adding data
| * 4ed5e08 fichier 3
| * 82b57e3 fichier 2
| * 09c403e fichier 1
|/
*   69620a8 Merge remote
```

```
* d3966a6 output2
* 2b4181a output1
* 4ed5e08 fichier 3
* 82b57e3 fichier 2
* 09c403e fichier 1
| * 79f5ba4 data2
| * d625223 data 1
| * 4a51090 Adding data
|/
*   69620a8 Merge remote
```

NEVER use rebase on commits already pushed

Quizz : Why ?

Rebasing (3)

```
* 5e3ef1b output2
* 29ee50d output1
| * 79f5ba4 data2
| * d625223 data 1
|/
* 4a51090 Adding data
| * 4ed5e08 fichier 3
| * 82b57e3 fichier 2
| * 09c403e fichier 1
|/
*   69620a8 Merge remote
```

```
* d3966a6 output2
* 2b4181a output1
* 4ed5e08 fichier 3
* 82b57e3 fichier 2
* 09c403e fichier 1
| * 79f5ba4 data2
| * d625223 data 1
| * 4a51090 Adding data
|/
*   69620a8 Merge remote
```

NEVER use rebase on commits already pushed

Quizz : Why ?

Creates other commits, which are in fact the sames.

OK for cleaning your work though.

Working with others

1. Check for whitespaces errors : `git diff --check`
2. Each commit should be about a specific change. Don't forget you can split your work into several commits ! More readable and easier for reverting.
3. Format your commit message. Ideally, a short description on the first line, followed by a body.

Working with others (2)

```
commit a3347b988a338e95b7f3b11eda776ac0d7b84dd0
Author: Linus Torvalds <torvalds@linux-foundation.org>
Date:   Fri May 25 09:02:03 2012 -0700

fmt-merge-message: add empty line between tag and signature
verification

When adding the information from a tag, put an empty line between
the message of the tag and the commented -out signature
verification information.

At least for the kernel workflow, I often end up re-formatting the
message that people send me in the tag data. In that situation,
putting the tag message and the tag signature verification back-to-
back then means that normal editor "reflow parapgraph" command will
 get confused and think that the signature is a continuation of the
 last message paragraph.

ETC...
Signed-off-by: Linus Torvalds <torvalds@linux-foundation.org>
Signed-off-by: Junio C Hamano <gitster@pobox.com>
```

Sharing

Patch

Contains differences you have added to the project. Two strategies :

- `git format-patch -M origin/master`
- `git diff --no-prefix > patchfile`

Applying patches

- `git apply mypatch.patch` if generated by `git diff`
- `git am mypatch.patch` if generated by `git format-patch`

Code

- `git archive master | gzip > myarchive.tar.gz` : archive with the code
- `git shortlog master` : create a summary of the commits

Tips

- `git bundle create myrepo.bundle master`
- `git clone myrepo.bundle -b master myrepo2`

Expert : Older commits

Refering to a commit :

- `commit 08920095740df4838b2b174624277c6372b40f3c`.
- `commit 089200957` (8-10 numbers should be enough).

Refering to a parent :

- `HEAD^` if several parents (merge)
- `HEAD~` if several parents (merge)

For example, the grandparent is `HEAD~2`.

Branches relations

- `git log master..donkey` : everything on donkey that is not in master :
- `git log origin/master..HEAD` : what I am about to commit

Negations :

- `git log ^donkey` : everything not on donkey. (or `git log --not donkey`)
- `git log master...donkey` : everything on each that is not common to the two.

Tips

You can see the older positions of head with `git reflog`. Warning : only since the cloning or for the last few months.

Detached head (ouch)

```
$ git checkout 2b4181a17
Note: checking out '2b4181a17'.

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 2b4181a... output1
```

```
$ git branch
* (no branch)
  master
  testing
  waiting
```

Detached head (2)

Walkthrough

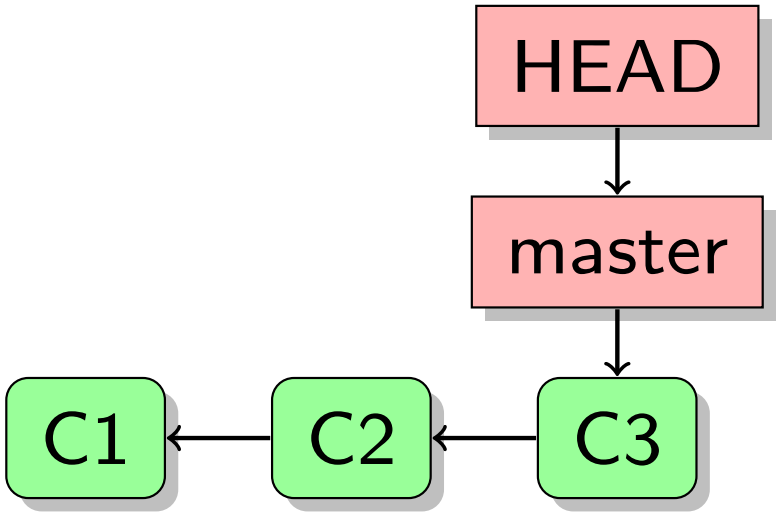
- `git checkout master`
- `git checkout 2b4181a17 -b temp`
- `git branch -f master temp` : make master point to temp
- `git checkout master`
- `git branch -d temp`
- `git push origin master`

Detached head (3)

Quizz : Why ?

Detached head (3)

Quizz : Why ?



Tips

Roll back a file : `git reset 24c47abf config.tex`

Add `--hard` if needed.

Interactive staging

```
$ git add -i
           staged     unstaged path
  1:         +0/-0      nothing interact1.txt
  2:         +0/-0      nothing interact2.txt

*** Commands ***
  1: status       2: update       3: revert       4: add untracked
  5: patch        6: diff         7: quit         8: help
What now> 4
  1: gitdemo2/
  2: tmp.txt
Add untracked>> 1
* 1: gitdemo2/
  2: tmp.txt
Add untracked>> 2
* 1: gitdemo2/
* 2: tmp.txt
Add untracked>>
Ignoring path gitdemo2/
added 2 paths

What now> q
Bye.
```

Stashing

Allows to temporary save your for switching to another tasks (without committing). Walkthrough :

- `git stash` : save
- `git checkout -b testing` : change to a new branch
- `git checkout master` : return back to master
- `git stash apply` : load saved data

We can also create a branch from a stashing :
`git stash branch donkey`

Tips

If several stash, `git stash list` and `git stash apply stash@{2}`.

Unapply stash : `git stash show -p stash@1 | git apply -R`

Rewriting the commits

Forgot to add a file or ashamed of your last commit message ?

`git commit --amend`

Never use it after a push

For rewriting several commits, `git rebase -i`

Golden rule :

It is okay to rewrite the history but only if you did not shared it.

Rewriting the commits

```
$ git rebase -i HEAD~3
pick 4ed5e08 fichier 3
pick 2b4181a output1
pick 15db31d New files
# Rebase 82b57e3..15db31d onto 82b57e3
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Rewriting the commits (2)

Note : the commits are in reverse order !

- pick : use the commit
- edit : waits for the user to change the commit. You will need to use `git commit --amend` and `git rebase --continue`
- squash : squashing two commits in one (will ask you to change the message)

What if you made a mistake ?

```
$ git rebase -i HEAD~3
Cannot 'squash' without a previous commit
$ git rebase --continue
cat: /home/alex/gitdemo/.git/rebase-merge/stopped-sha: No such file
 or directory
Cannot 'squash' without a previous commit
$ git rebase --abort
```

Rewriting the commits (3)

You can also :

- reorder the commits
- add new commits

Example :

```
$ git rebase -i HEAD~3
pick 4ed5e08 fichier 3
edit 2b4181a output1
pick 15db31d New files
$ git add uselessfile.txt
$ git commit -m "Adding an useless file"
$ git rebase --continue
```

Of course, don't forget to merge to master !

Rewriting the commits (4)

Tips

Overkill : Remove the file from all history and on all branches.

```
git filter-branch --tree-filter -all 'rm -f confidential.dat' HEAD
```

Avoid it if you want to keep your friends..

Who broke the code ?

`git blame` show the last modification of a file.
Here `^` means the line hasn't been edited since the first commit.

```
$git blame -L 7,9 adv_u_alone.m
4a97aede matlab/adv_u_alone.m (Alex [...])     function [delta_f,
                        delta_f1] = variation_C(i,j,q,dq,dlon,dt,n)
^2d33bc0 matlab/adv_u_alone.m (Alex [...])         if (j > 1)
^2d33bc0 matlab/adv_u_alone.m (Alex [...])           j_prev = j-1
```

With the `-C` option, Git searches where the code came from (if copied).

Who broke the code ? (2)

You can use a binary search for finding the faulty commit :

```
$ git bisect start
$ git bisect bad
$ git bisect good 7e7ddf1d3
Bisecting: 2 revisions left to test after this (roughly 2 steps)
[08d837f9860aae96ce7793e35e8eb16e261faaa3] Expert mode
$ git bisect good
Bisecting: 0 revisions left to test after this (roughly 1 step)
[1976699eea79323132a60af6707f917d75796ee0] Rebase finally done. Git
 bundle.
$ git bisect reset
```

Git inside Git

Add another Git repository inside yours.

```
$ git submodule add git@github.com:alexDarcy/scripts.git
myscripts
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# new file:   .gitmodules
# new file:   myscripts
#
$ git commit -am "Commiting the submodule"
```

Inside myscripts : Git workflow as usual. Outside : the submodule is treated as a commit.

Git inside Git

Beware, cloning the project is tricky

```
$ git clone git@github.com:alexDarcy/gitdemo.git bisdemo
$ cd bisdemo/
$ git submodule init
$ git submodule update
```

Warning : `update` will not put you on a branch, you must checkout to master (or create a branch) !

Another trick : if code is commited inside the submodule, and then on the global directory

```
$ git pull
  Fast-forward
   myscripts |    2 +-
  1 files changed, 1 insertions(+), 1 deletions(-)
$ git submodule update
  From github.com:alexDarcy/scripts
    87cf769..64ddea6  master     -> origin/master
    Submodule path 'myscripts': checked out
   '64ddea6775c70209262d3f5ba4d91fb6ec3348a6'
```

Git inside Git 2.0

Another way for the same problem : subtree.

```
$git remote add scripts_remote git@github.com:alexDarcy/scripts.
git
$ git fetch scripts_remote
$ git checkout -b scripts_branch scripts_remote/master
Branch scripts_branch set up to track remote branch master from
scripts_remote.
Switched to a new branch 'scripts_branch'
$ git checkout master
Switched to branch 'master'
$ git read-tree --prefix=scripts/ -u scripts_branch
```

Now, your directory scripts contains the repository. You can update it with

```
$ git checkout scripts_branch
Switched to branch 'scripts_branch'
$ git pull
$ git checkout master
Switched to branch 'master'
$ git merge --squash -s subtree --no-commit scripts_branch
Squash commit -- not updating HEAD
Automatic merge went well; stopped before committing as requested
```

For a better Git experience

Ignoring files

With a .gitignore file in your repository, you can tell Git to ignore some files.

```
*.a
!lib.a
build/
doc/*.txt
```

Or use a global file :

```
git config --global core.excludesfile ~/.gitignore_global
```

Customization

- Colors : `git config --global color.ui true`
- Aliases : `git config --global alias.co checkout`

Other Git possibilities

- Huge configuration possibilities : template for commit messages, customized editor, external diff tools...
- Diff on binary files
- Keyword substitution
- Different merge strategies
- Hooks (running tests, generating documentation...)
- Bridge to a SVN server
- External tools for migrating to Git from : SVN, Perforce, CVS (or write your own)
- Tags

Another cool feature...

You can diff Word documents :

- Put in your `.gitattributes` file, the following `*.doc diff=word`
- Add a tool for converting Word to text :

```
$ git config diff.word.textconv strings
```

Another cool feature (2)

The difficulty is to find a good Word-to-text converter.
Here I used `catdoc` on a Word 2007 document :

```
$ git diff document.doc
diff --git a/document.doc b/document.doc
index 2bf139c..71c1dd8 100644
--- a/document.doc
+++ b/document.doc
@@ -2,3 +2,7 @@ Hello,

 I am testing.

 +Here, a new line.
 +
 +And a new paragraph.
```

You can use this for .odf documents or jpg images (with the `exiftool`).

Troubleshooting

`Cannot rewrite branch(es) with a dirty working directory`

If you changed branches without commiting, you will end it with a "dirty" working dir. Either commit the changes or stash them.

Getting help

- the `man` : `man git-log`
- the official documentation (inspired most of this presentation)
- Google.

Thank you for your attention.