Setup

- Install git
  - Ubuntu: `sudo apt-get install git`
  - Fedora: `sudo yum install git`
  - Windows: `http://msysgit.github.io/`
- Create a `https://github.com/` account
- `git config --global user.name "Badger Blaireau"`[1]
- `git config --global user.email "badger@blaireau.com"`
- `git config --global color.ui true`
- `git config --global credential.helper cache`

[1] Use your actual name and email

UrLab: Git workshop

Guillaume Desmottes (gdesmott@gnome.org)

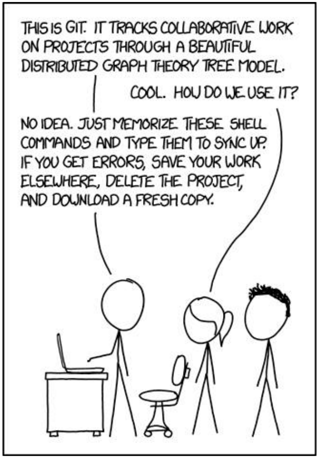
16th November 2015

Plan

About Me

- Guillaume Desmottes
- Free and Open Source hacker
- Empathy maintainer, Telepathy developer
- GStreamer developer
- `http://blog.desmottes.be`
- *@gdesmott*

Git


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Goals

- Get to know Git
- Basic operations
- Demystify the beast

Organization

- Theory + exercices
    - Command line

Plan

Why use a VCS?

- History
- Avoid loosing data
- Sort your WIP work
- Sync between devices
- Share with others

Git

- Distributed version control system
- Started by Linus Torvalds
- Non-linear development
- Designed like a FS
- Efficient, scalable and fast
- Very powerful

Centralized Version Control System

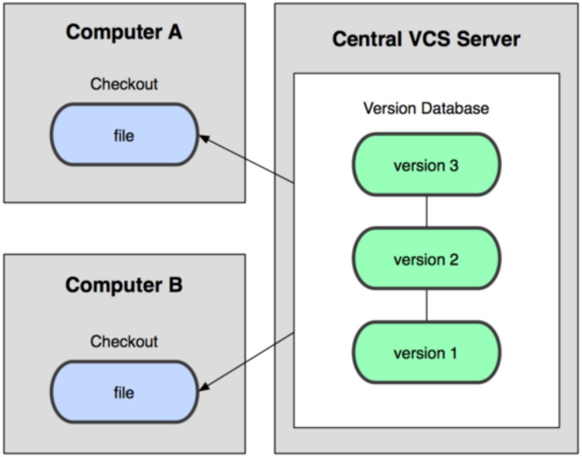


Figure: Centralized Version Control System (© Pro Git)

Distributed Version Control System

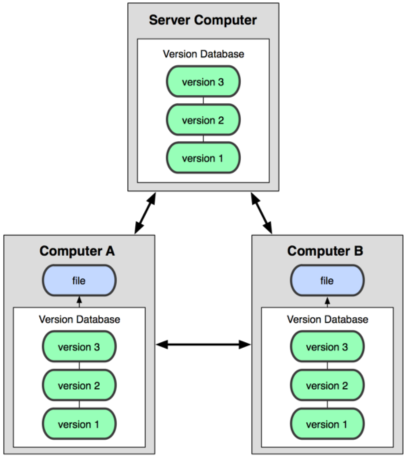


Figure: Distributed Version Control System (© Pro Git)

Config

- `git config --global user.name "Badger Blaireau"`
- `git config --global user.email "badger@blaireau.com"`
- `git config --global color.ui true`
- `git config --global core.editor gedit`
- `git config --global credential.helper cache`

Create your first repo

- Create a directory *badger* (`mkdir`)
- Go to this directory (`cd`)
- Create test.txt with some content
- `git init`
- `git add test.txt`
- `git commit -m "my first commit"`
- `git show`

Clone an existing repo

- `https://github.com/gdesmott/nice-words`

Clone an existing repo

- `https://github.com/gdesmott/nice-words`
- `git clone https://github.com/gdesmott/nice-words.git`

Commit

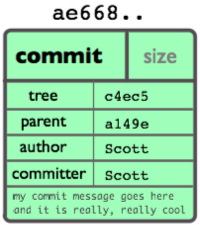


Figure: commit (© Git Community Book)

- links a physical state of a tree
- describs of how we got there and why
- build a graph

SHA

- 0a6ce2b0c136c05fa5395d51517208c041bc392d
- 40-digit ”object name”
- SHA-1 of the object
- Fast comparaison
- Persistent naming
- Error proof

Commit

```
$ git show --pretty=raw
commit 0a6ce2b0c136c05fa5395d51517208c041bc392d
tree 27340d15c639a9b600c1ce1c452cc9d0e84f396f
parent b850f8c1c6106ac50b26e090c11bd1c7ae8c0f9d
author Guillaume Desmottes <guillaume.desmottes@collabora.co.uk> 1305108893 +0200
committer Guillaume Desmottes <guillaume.desmottes@collabora.co.uk> 1305108986 +0200

    theme_adium_remove_focus_marks: early return if there is no unread message
```

Multiple commits

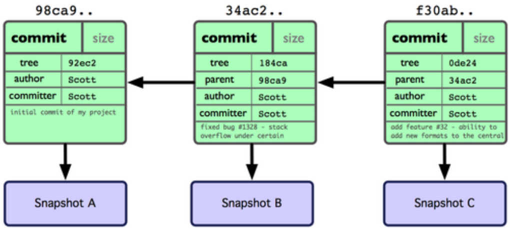


Figure: Multiple commits (© Pro Git)

Commit

- set of changes
- commit message (short, long)
- `git commit -am "Some cool changes"`
-
```
commit d5f33d45e4c0e306e8d16b4573891a65d9ad544f
Author: Axel Lin <axel.lin@gmail.com>
Date:   Tue May 17 15:44:09 2011 -0700

    drivers/leds/leds-lm3530.c: add MODULE_DEVICE_TABLE

    Adding the necessary MODULE_DEVICE_TABLE() information allows the driver
    to be automatically loaded by udev.
```

Staging Area

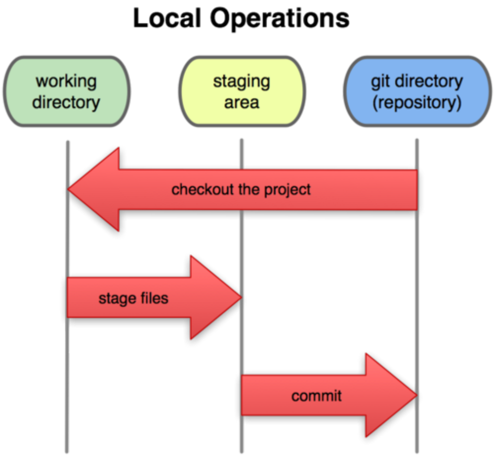


Figure: Working directory, staging area, and git directory. (© Pro Git)

Select what we want to commit

- `git status`
- `git add <file>`
- `git add -p`

Good practice

- Commit early, commit often
- Atomic commits
- Do **not** break build/tests
- Do **not** depend on newer commits

Exercices

- Use the `nice-words` repo
- Append one word to *animals.txt* and one to cities.txt
- `git diff`
- Commit everything (`commit -a`)
- Append 5 words to *animals.txt* and one to cities.txt
- Commit file by file (`status`, `add`)
- **Prepend** one word and **append** one to *animals.txt*
- Commit each word (`status`, `add -p`)
- `git log`
- `git show`

Branches

- lightweight pointers to a commit
- updated automatically
- `master`
- `git branch`
- `git checkout`

master

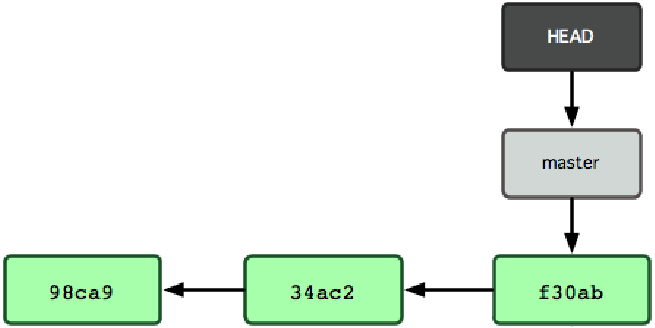


Figure: Working in the master branch. (© Pro Git)

branching

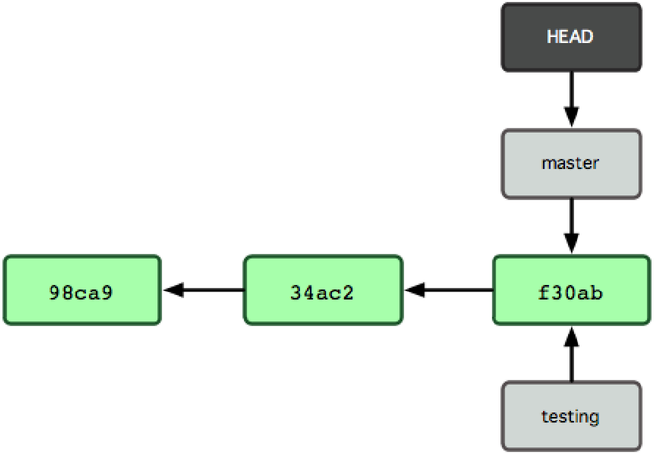


Figure: Branching `testing`. (© Pro Git)

checking out

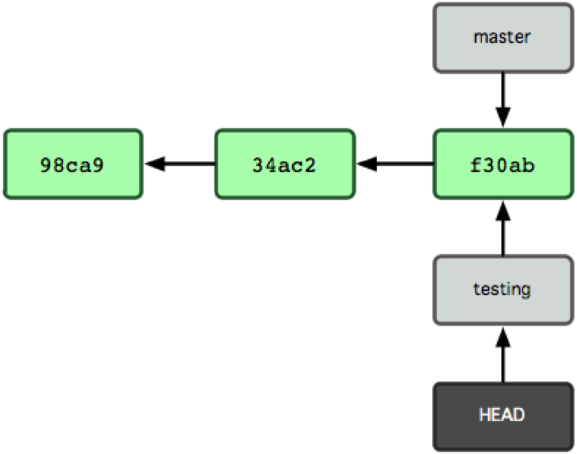


Figure: checkout `testing`. (© Pro Git)

commiting

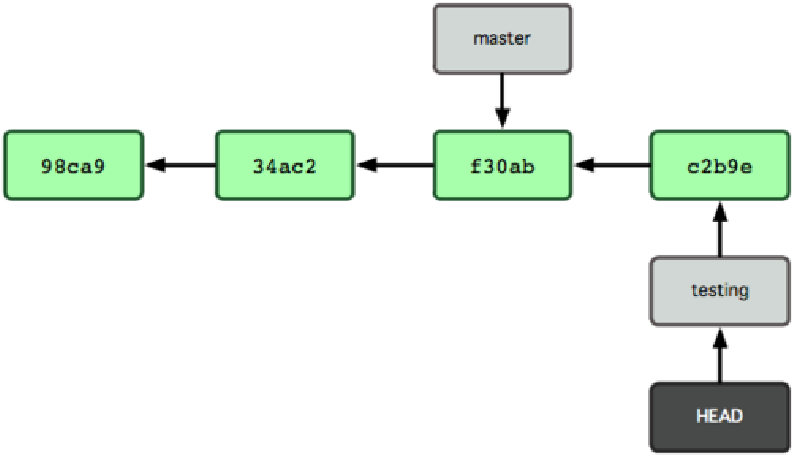


Figure: commiting in `testing`. (© Pro Git)

checking out

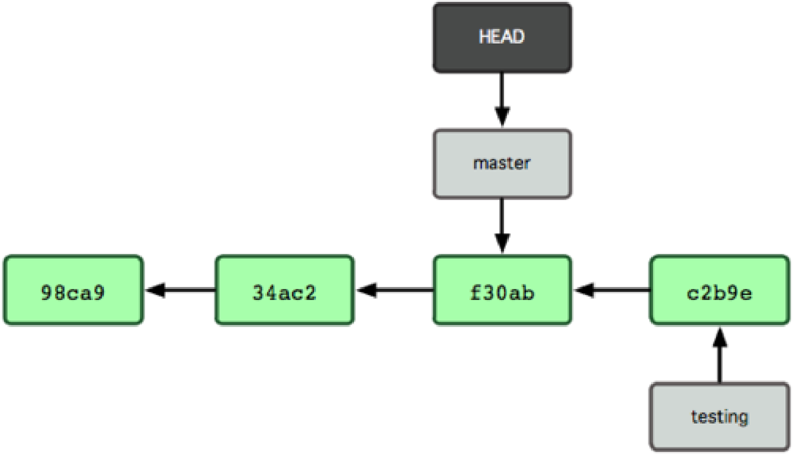


Figure: checkout `master`. (© Pro Git)

commiting

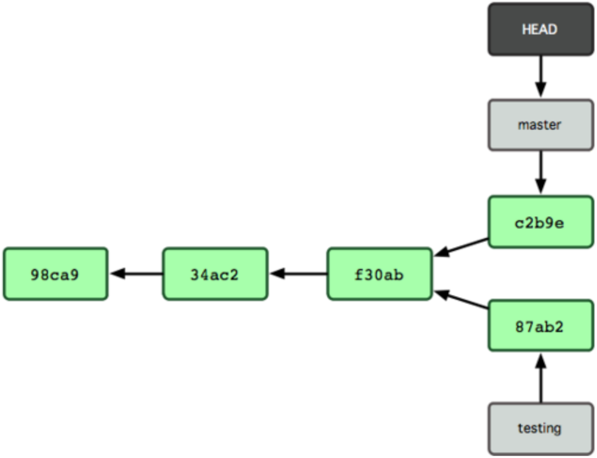


Figure: commiting in `master`. (© Pro Git)

Good practice

- Branches are cheap!
- **Always** work in branches
- Keep `master` clean
- Work in feature/bug branches
- Properly name branches: `video-widget-race-603588`
- Avoid monster branches

Exercices

- (`git checkout master && git reset --hard origin/exo1`)
- Create and checkout a branch *farm* (`branch, checkout`)
- Add some changes in *animals.txt* and commit
- Create and checkout a branch *fish* **based on master**
- Add some changes in *animals.txt* and commit
- `git log farm`
- `git log fish`
- Create and checkout a branch *europe* **based on master**
- Add some changes in cities.txt and commit
- Create and checkout a branch *birds* **based on master**
- Add some changes in *animals.txt* and commit

Merge

- Join two or more development histories together
- Clean working tree
- `git merge`

Fast-forward merge

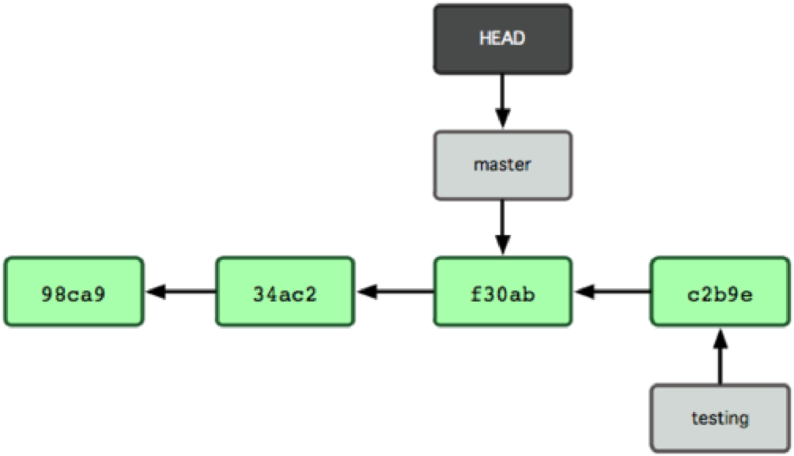


Figure: `master` did not diverge. (© Pro Git)

Fast-forward merge

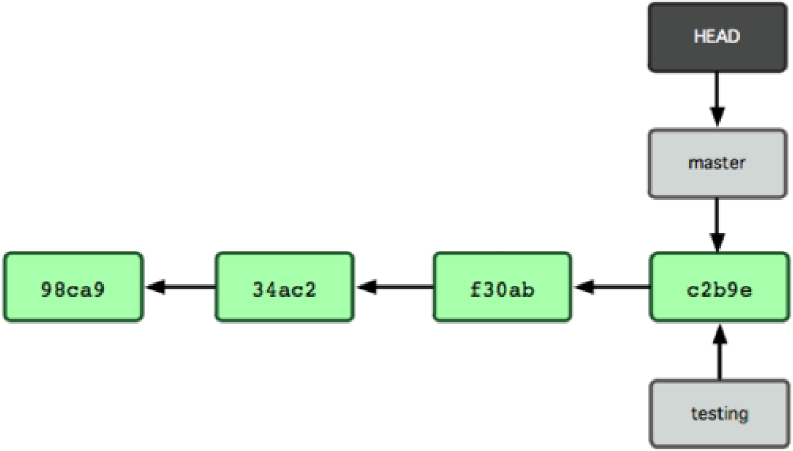


Figure: fast-forward merge of `testing` into `master`. (© Pro Git)

Merge

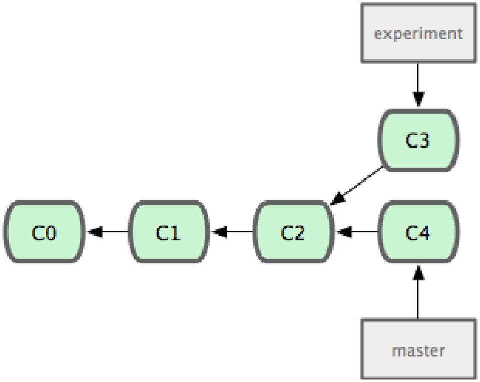


Figure: diverged commit history. (© Pro Git)

Merge

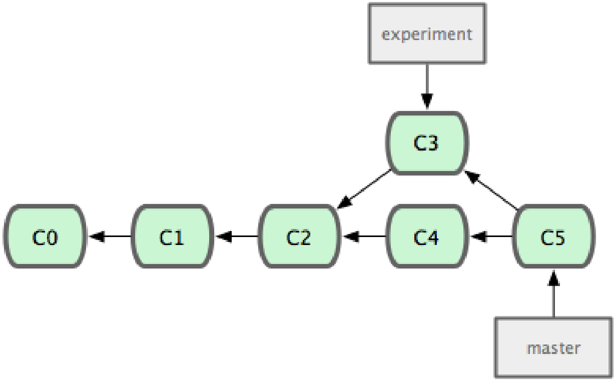


Figure: merge introducing a merge commit. (© Pro Git)

Exercices

- Merge branch *farm*[2] into *master*
  - Checkout `master`
  - `merge farm`
- `git log`
- Merge branch *europe* into *master*
- `git log`
- `git log --graph --all --decorate --oneline`

[2] origin/farm

Conflicts

- Try merging a branch

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

Conflicts

- Can't commit any more

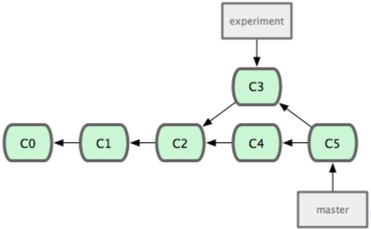
```
$ git commit
file.txt: needs merge
```

```
$ git status
file.txt: needs merge
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working dir
#
# unmerged:    file.txt
```

Conflicts

- Don’t panic!
- Solve the conflict

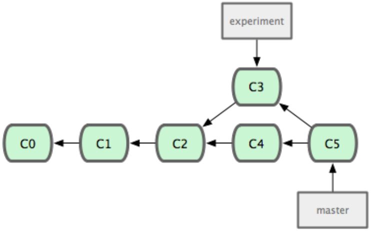
```
$ git diff
diff --cc file.txt
index 557db03,2b60207..0000000
--- a/file.txt
+++ b/file.txt
@@@ -1,1 -1,1 +1,5 @@@
++<<<<<<< HEAD
 +Hello World
++=======
+ Goodbye
++>>>>>>> experiment
```



Conflicts

- Solve the conflict

```
$ git diff
diff --cc file.txt
index 557db03,2b60207..0000000
--- a/file.txt
+++ b/file.txt
@@@ -1,1 -1,1 +1,1 @@@
- Hello World
 -Goodbye
++Goodbye World
```



Conflicts

- Add the resolved files to the stage
- Commit

Conflicts: resolve

- `git checkout --ours <file>`
- `git checkout --their <file>`

Exercices

- (`git checkout master && git reset --hard origin/exo3`)
- Try merging *origin/fish* into *master*
- Identify the conflict (`status`)
- Fix it! (`add`)
- Finish the merge (`commit`)

Push

- Publish your changes
- Need your own repo
- https://github.com/gdesmott/nice-words $\rightarrow$ Fork
- `git remote add my-repo https://github.com/$USERNAME/nice-words.git`
- Checkout *master*
- `git push my-repo master`

Push; non-fast-forward

```
 git push origin master
To /home/cassidy/dev/test-git/upstream.git/
! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to '/home/cassidy/dev/test-git/upstream.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.
```

Problems with merging

- Potentially lot of conflicts to deal with
- Lot of merge commits
- every committer for a time has responsibility for what the other committers have committed
- Can end up in disasters if developers don’t merge properly
- Confuse `git bisect`

Rebase

- Forward-port local commits to the updated upstream head
- `git rebase`

Rebase

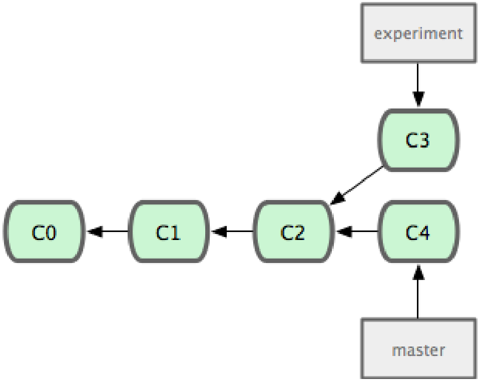


Figure: diverged commit history. (© Pro Git)

Rebase

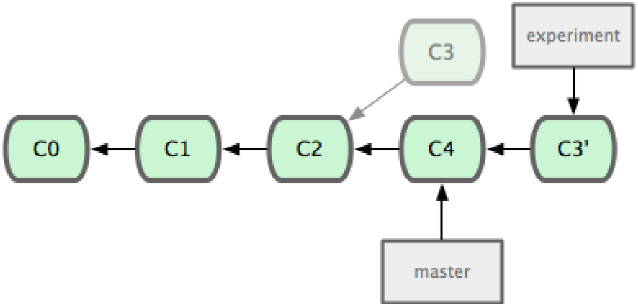


Figure: `C3` has been rebased on top of `master`. (© Pro Git)

Rebase

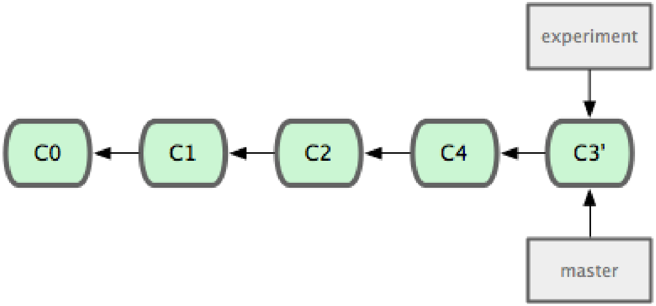


Figure: Fast-forward `master`. (© Pro Git)

Advantages

- Developper takes responsability that his work applies on top of master
- Keeps a clean history
- More similar to CVS/SVN's workflow
- Gives a chance to clean up the branch before pushing

Cons

- A bit more complex
- Be **very** careful to not rewrite public history !
  - **public** branch: more than one person pulls from.
  - **topical** branch (or **feature** branch): private branch

Exercices

- (`git checkout master && git reset --hard origin/exo4`)
- Rebase *birds* on top of *master*
  - checkout *birds*
  - rebase *master*
  - Fix conflict (`add`, `rebase --continue`)
- Merge *birds* into *master*
- `git log`

Stash

- Stash the changes in a dirty working directory away
- `git stash`
- Avoid to record half-done commit
- `apply` / `pop`

Stash

- Before `pulling`
- Can be unstashed on top of another commit
- Conflicts
    - not conflicting file added to to stage
    - resolve conflict + stage
    - commit