Git

Or Git as a true programmer

Thibaut SAUTEREAU (thithib)

November 22, 2016

Telecom SudParis

0x00 Introduction

Introduction

- Version control system used for software development

Introduction

- Version control system used for software development
- Created by Linus Torvalds in 2005 for Linux

Introduction

- Version control system used for software development
- Created by Linus Torvalds in 2005 for Linux
- Free (GPLv2)

Introduction

- Version control system used for software development
- Created by Linus Torvalds in 2005 for Linux
- Free (GPLv2)
- Runs on Linux, Windows and OS X

Introduction

- Version control system used for software development
- Created by Linus Torvalds in 2005 for Linux
- Free (GPLv2)
- Runs on Linux, Windows and OS X
- Distributed (unlike CVS and SVN) and much more powerful than Mercurial, but a bit harder to master

Outline

- Git basics

Outline

- Git basics
- Branches

Outline

- Git basics
- Branches
- Getting powerful

0x01 Git basics

Set up your name and e-mail address

- They will be referenced in each of your commits

Set up your name and e-mail address

- They will be referenced in each of your commits
- `git config --global user.name 'My Name'`

Set up your name and e-mail address

- They will be referenced in each of your commits
- `git config --global user.name 'My Name'`
- `git config --global user.email me@mydomain.net`

Starting a project

- `git init <repository>` → create a new repository

Starting a project

- `git init <repository>` → create a new repository
- `git clone <url>` → clone an existing remote repository

Starting a project

- `git init <repository>` → create a new repository
- `git clone <url>` → clone an existing remote repository
- `git://` is a special Git protocol. Most repositories can also be accessed using `http://` but this is slower.

Starting a project

```
thithib@cronos:/tmp$ git init test
Dépôt Git vide initialisé dans /tmp/test/.git/

thithib@cronos:/tmp$ cd test

thithib@cronos:/tmp/test master$ ls -la
total 0
drwx------  3 thithib users  60 18 oct.  00:39 .
drwxrwxrwt 11 root    root  260 18 oct.  00:39 ..
drwx------  7 thithib users 200 18 oct.  00:39 .git

thithib@cronos:/tmp/test master$ git remote add origin git@bitbucket.org:thithib/test.git

thithib@cronos:/tmp/test master$ cd .. && rm -r test

thithib@cronos:/tmp$ git clone git@bitbucket.org:thithib/test.git
Clonage dans 'test'...
Enter passphrase for key '/home/thithib/.ssh/id_rsa':
warning: Vous semblez avoir cloné un dépôt vide.

thithib@cronos:/tmp$ cd test && ls -la
total 0
drwx------  3 thithib users  60 18 oct.  00:39 .
drwxrwxrwt 11 root    root  260 18 oct.  00:39 ..
drwx------  7 thithib users 200 18 oct.  00:40 .git
```

What am I doing?

- `git status` → show edited/new/removed files

What am I doing?

```
thithib@cronos:/tmp/test master$ vim hello.c

thithib@cronos:/tmp/test master []$ cat hello.c
#include <stdio.h>

int main(void)
{
        puts("hello, world");
        return 0;
}

thithib@cronos:/tmp/test master []$ git status
Sur la branche master

Validation initiale

Fichiers non suivis:
  (utilisez "git add <fichier>..." pour inclure dans ce qui sera validé)

        hello.c
```

Committing

- `git add <file>` → stage file for next commit

Committing

- `git add <file>` → stage file for next commit
- `git commit [-m "<message>"]` → commit

Committing

- `git add <file>` → stage file for next commit
- `git commit [-m "<message>"]` → commit
- `git log [-p] [<file>]` → show commits history

Committing

- `git add <file>` → stage file for next commit
- `git commit [-m "<message>"]` → commit
- `git log [-p] [<file>]` → show commits history
- `git diff` → show differences since last commit

Committing

```
thithib@cronos:/tmp/test master []$ git add hello.c

thithib@cronos:/tmp/test master []$ git status
Sur la branche master

Validation initiale

Modifications qui seront validées :
  (utilisez "git rm --cached <fichier>..." pour désindexer)

        nouveau fichier : hello.c


thithib@cronos:/tmp/test master []$ git commit -m "Initial hello world"
[master (commit racine) 3fb2490] Initial hello world
 1 file changed, 7 insertions(+)
 create mode 100644 hello.c
```

Committing

```
commit 3fb24906fa375c628009102dcf6f8ef984f93e98
Author: Thibaut SAUTEREAU <thibaut.sautereau@telecom-sudparis.eu>
Date:   Tue Oct 18 00:54:46 2016 +0200

    Initial hello world

diff --git a/hello.c b/hello.c
new file mode 100644
index 0000000..c19db98
--- /dev/null
+++ b/hello.c
@@ -0,0 +1,7 @@
+#include <stdio.h>
+
+int main(void)
+{
+        puts("hello, world");
+        return 0;
+}
(END)
```

Committing

```
diff --git a/hello.c b/hello.c
index c19db98..b861a41 100644
--- a/hello.c
+++ b/hello.c
@@ -1,7 +1,8 @@
 #include <stdio.h>
+#include <stdlib.h>

 int main(void)
 {
         puts("hello, world");
-        return 0;
+        return EXIT_SUCCESS;
 }
(END)
```

Sending my work

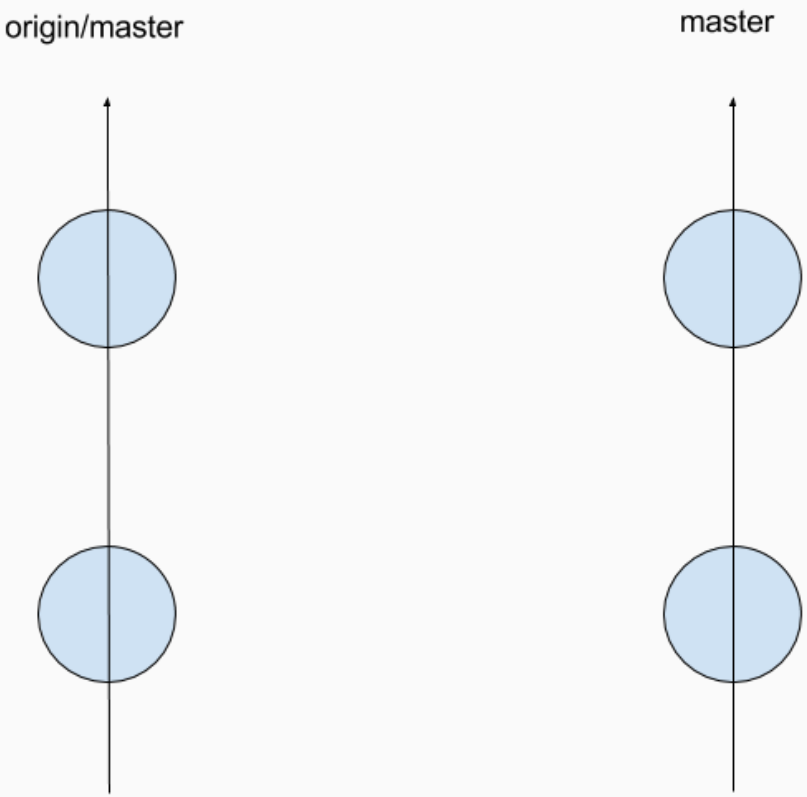
- `git push` $\rightarrow$ update remote refs along with associated objects

Sending my work

```
thithib@cronos:/tmp/test master$ git status
Sur la branche master
Votre branche est basée sur 'origin/master', mais la branche amont a disparu.
  (utilisez "git branch --unset-upstream" pour corriger)
rien à valider, la copie de travail est propre

thithib@cronos:/tmp/test master$ git push
Warning: Permanently added the RSA host key for IP address '2401:1d80:1010::151' to the list of known hosts.
Enter passphrase for key '/home/thithib/.ssh/id_rsa':
Décompte des objets: 6, fait.
Delta compression using up to 4 threads.
Compression des objets: 100% (4/4), fait.
Écriture des objets: 100% (6/6), 604 bytes | 0 bytes/s, fait.
Total 6 (delta 1), reused 0 (delta 0)
To bitbucket.org:thithib/test.git
 * [new branch]      master -> master
```

Current branches



Syncing with others - Merge VS Rebase

- `git fetch <remote>` $\rightarrow$ fetch branches from remote repository

Syncing with others - Merge VS Rebase

- `git fetch <remote>` → fetch branches from remote repository
- `git rebase <onto> [<branch>]` → replay commits of two branches from last commit in common

Syncing with others - Merge VS Rebase

- `git fetch <remote>` → fetch branches from remote repository
- `git rebase <onto> [<branch>]` → replay commits of two branches from last commit in common
- `git merge [--no-ff] <branch>` → merge two diverging branches, in a new commit or not

Divergence

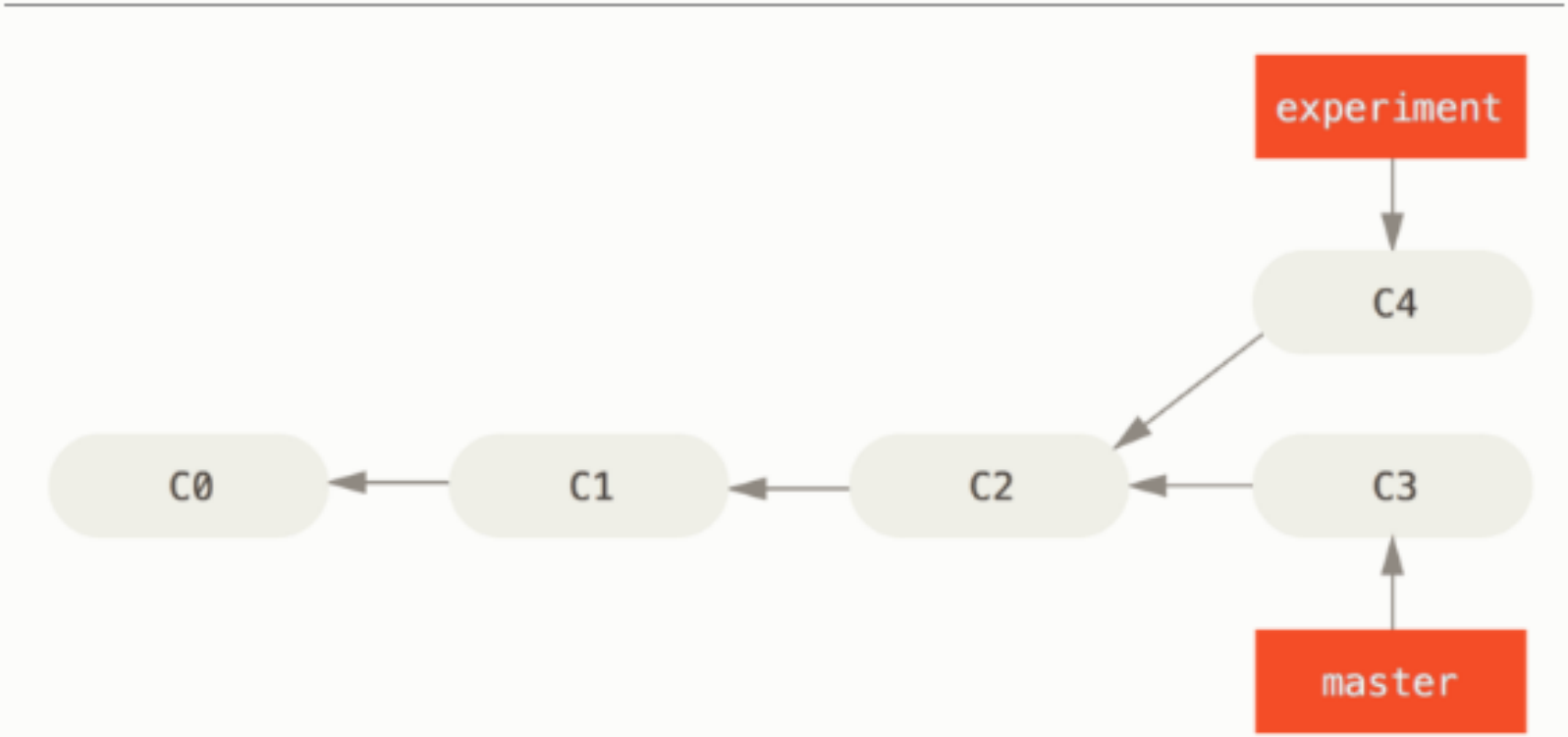


Figure 3-27. Simple divergent history

(True) Merge

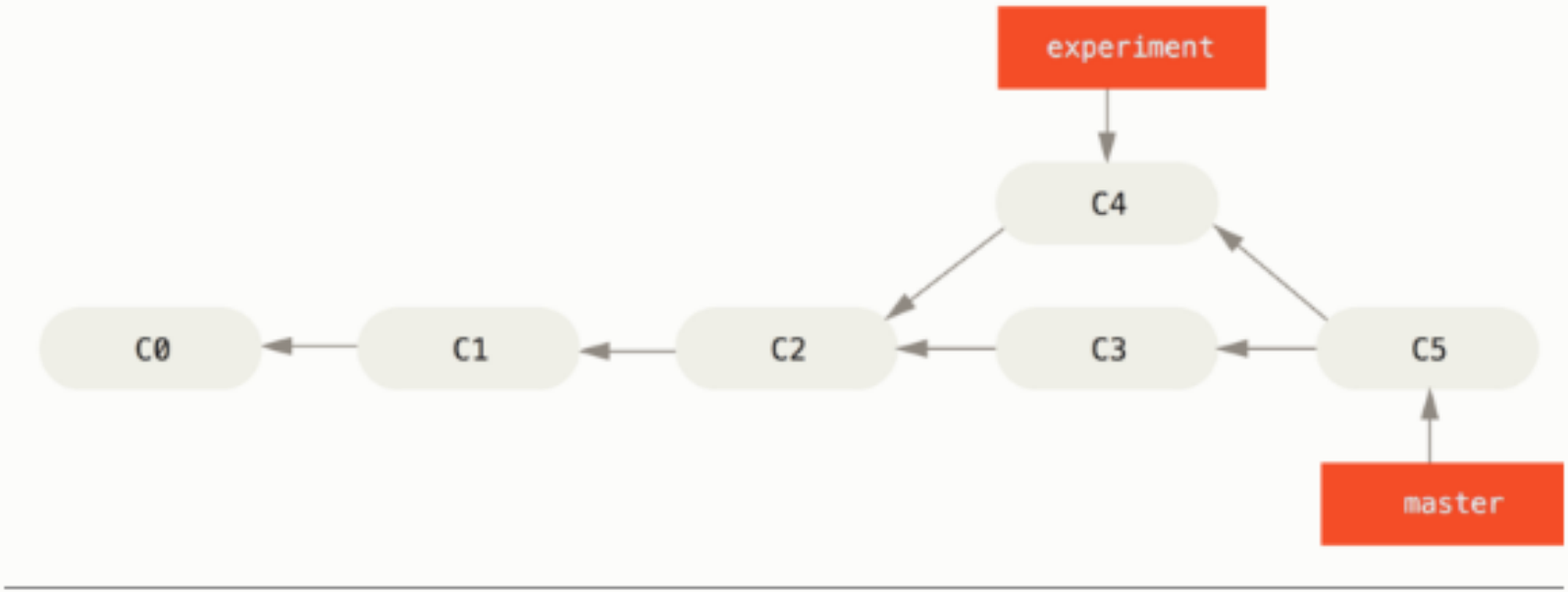


Figure 3-28. Merging to integrate diverged work history

Rebase

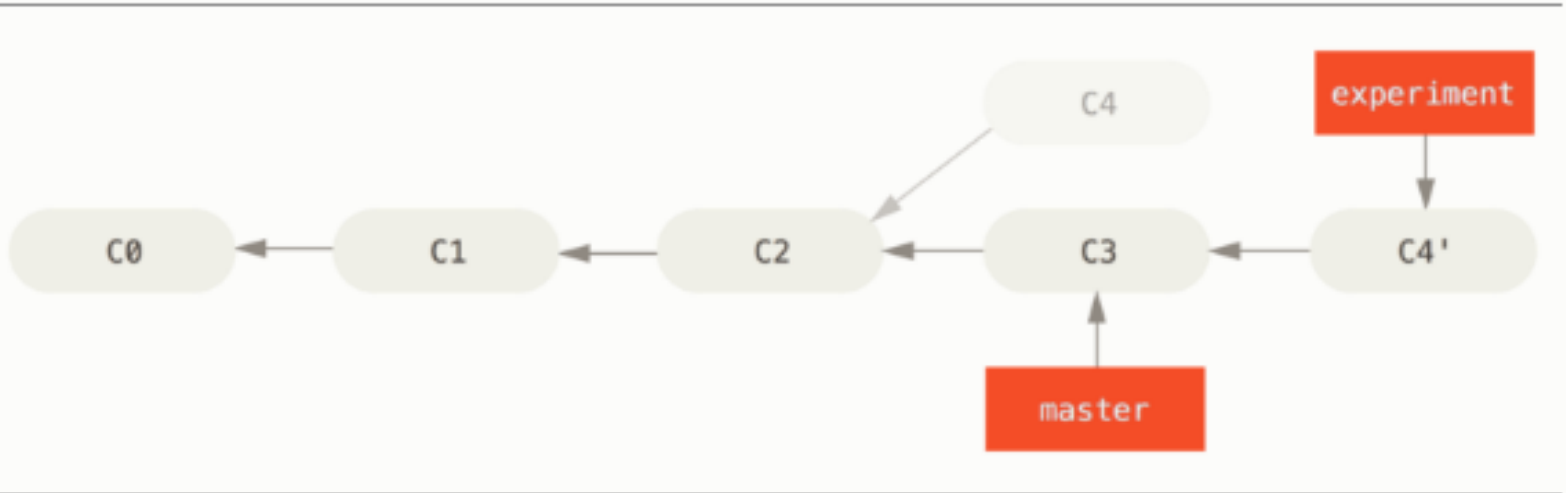


Figure 3-29. Rebasing the change introduced in `C4` onto `C3`

Live demo

Live demo

Merge VS Rebase - When?

- Merge:

Merge VS Rebase - When?

- Merge:
  - local or temporary branch: *fast-forward merge*

Merge VS Rebase - When?

- Merge:
  - local or temporary branch: *fast-forward merge*
  - true branch with own meaning: *true merge*

Merge VS Rebase - When?

- Merge:
  - local or temporary branch: *fast-forward merge*
  - true branch with own meaning: *true merge*
- Rebase:

Merge VS Rebase - When?

- Merge:
  - local or temporary branch: *fast-forward merge*
  - true branch with own meaning: *true merge*
- Rebase:
  - when my push is refused because someone sent new work: rebase on remote branch so I don't pollute graph with micro-merges!

Merge VS Rebase - When?

- Merge:
  - local or temporary branch: *fast-forward merge*
  - true branch with own meaning: *true merge*
- Rebase:
  - when my push is refused because someone sent new work: rebase on remote branch so I don't pollute graph with micro-merges!
  - before doing a *fast-forward* merge

Merge VS Rebase - When?

- Merge:
  - local or temporary branch: *fast-forward merge*
  - true branch with own meaning: *true merge*
- Rebase:
  - when my push is refused because someone sent new work: rebase on remote branch so I don't pollute graph with micro-merges!
  - before doing a *fast-forward* merge
  - to clean my local history before sending my work (later)

0x02 Branches

Branches

- Git encourages the use of branches, including and especially local ones

Branches

- Git encourages the use of branches, including and especially local ones
- Whenever you start working on something, you should make a branch:

Branches

- Git encourages the use of branches, including and especially local ones
- Whenever you start working on something, you should make a branch:
  - It is local-only, nobody will see it

Branches

- Git encourages the use of branches, including and especially local ones
- Whenever you start working on something, you should make a branch:
  - It is local-only, nobody will see it
  - It is fast and easy

Branches

- Git encourages the use of branches, including and especially local ones
- Whenever you start working on something, you should make a branch:
  - It is local-only, nobody will see it
  - It is fast and easy
  - It allows you to split your work in different topics. Try something new, throw it away if it doesn't lead to something

Branches

- Git encourages the use of branches, including and especially local ones
- Whenever you start working on something, you should make a branch:
  - It is local-only, nobody will see it
  - It is fast and easy
  - It allows you to split your work in different topics. Try something new, throw it away if it doesn't lead to something
- Keep tracking branches *clean* at every moment so you can quickly pull new work from teammates when needed

Branches

- `git checkout -b <new-branch> <start-point>` → create local branch and check it out (oneshot)

Branches

- `git checkout -b <new-branch> <start-point>` → create local branch and check it out (oneshot)
- `git checkout <branch>` → move to `<branch>`

Branches

- `git checkout -b <new-branch> <start-point>` → create local branch and check it out (oneshot)
- `git checkout <branch>` → move to `<branch>`
- `git branch -r` → list remote-tracking branches

Branches

- `git checkout -b <new-branch> <start-point>` → create local branch and check it out (oneshot)
- `git checkout <branch>` → move to `<branch>`
- `git branch -r` → list remote-tracking branches
- `git push --set-upstream origin <branch>` → add upstream reference for current branch and push (only required for first push)

Branches

- `git checkout -b <new-branch> <start-point>` → create local branch and check it out (oneshot)
- `git checkout <branch>` → move to `<branch>`
- `git branch -r` → list remote-tracking branches
- `git push --set-upstream origin <branch>` → add upstream reference for current branch and push (only required for first push)
- `git push :<branch>` → delete remote branch

Branches

- `git checkout -b <new-branch> <start-point>` → create local branch and check it out (oneshot)
- `git checkout <branch>` → move to `<branch>`
- `git branch -r` → list remote-tracking branches
- `git push --set-upstream origin <branch>` → add upstream reference for current branch and push (only required for first push)
- `git push :<branch>` → delete remote branch
- `git branch -d <branch>` → delete local branch

Branches

- `git checkout -b <new-branch> <start-point>` → create local branch and check it out (oneshot)
- `git checkout <branch>` → move to `<branch>`
- `git branch -r` → list remote-tracking branches
- `git push --set-upstream origin <branch>` → add upstream reference for current branch and push (only required for first push)
- `git push :<branch>` → delete remote branch
- `git branch -d <branch>` → delete local branch
- `git branch -D <branch>` → delete local branch even if not merged

Branches

- `git checkout -b <new-branch> <start-point>` → create local branch and check it out (oneshot)
- `git checkout <branch>` → move to `<branch>`
- `git branch -r` → list remote-tracking branches
- `git push --set-upstream origin <branch>` → add upstream reference for current branch and push (only required for first push)
- `git push :<branch>` → delete remote branch
- `git branch -d <branch>` → delete local branch
- `git branch -D <branch>` → delete local branch even if not merged
- `git branch -m <old-branch-name> <new-branch-name>` → rename branch

0x03 Getting powerful

Miscellaneous

- `git checkout [<commit>] <path>` $\rightarrow$ reset file as in `<commit>`

Miscellaneous

- `git checkout [<commit>] <path>` → reset file as in `<commit>`
- `git rm [--cached] <path>` → remove file from working tree and from index

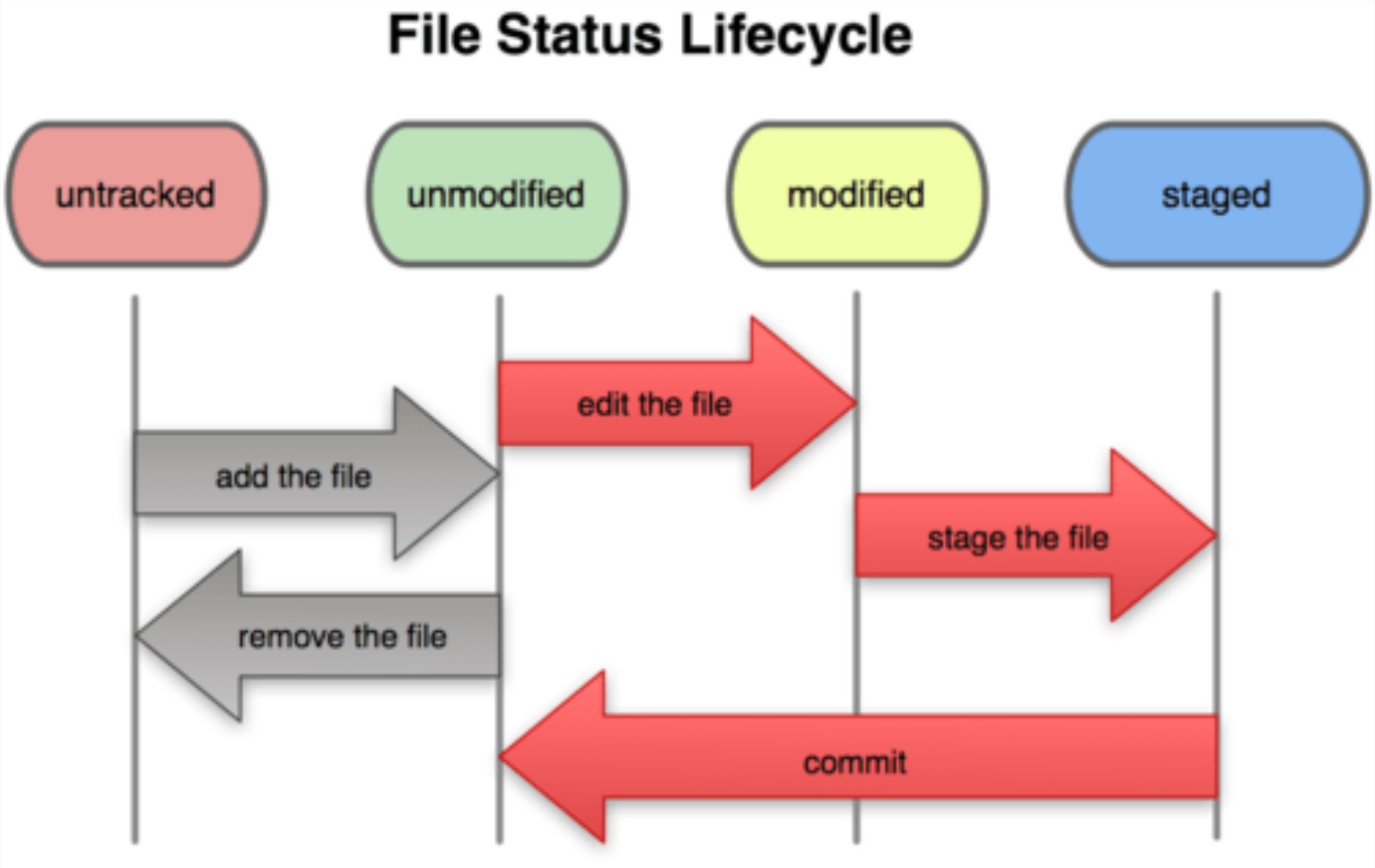
Miscellaneous

- `git checkout [<commit>] <path>` → reset file as in `<commit>`
- `git rm [--cached] <path>` → remove file from working tree and from index
- `git show <commit>` → show commit details

Miscellaneous

- `git checkout [<commit>] <path>` → reset file as in `<commit>`
- `git rm [--cached] <path>` → remove file from working tree and from index
- `git show <commit>` → show commit details
- `git revert <commit>` → create commit that reverts changes introduced by `<commit>`

File lifecycle



Stash

- `git stash` → stash the changes in a dirty working directory away

Stash

- `git stash` → stash the changes in a dirty working directory away
- `git stash list` → list the stashes that you currently have

Stash

- `git stash` → stash the changes in a dirty working directory away
- `git stash list` → list the stashes that you currently have
- `git stash show [-p] [<stash>]` → show the changes recorded in the stash as a diff

Stash

- `git stash` → stash the changes in a dirty working directory away
- `git stash list` → list the stashes that you currently have
- `git stash show [-p] [<stash>]` → show the changes recorded in the stash as a diff
- `git stash apply [<stash>]` → apply the stash on top of the current working tree state

Stash

- `git stash` → stash the changes in a dirty working directory away
- `git stash list` → list the stashes that you currently have
- `git stash show [-p] [<stash>]` → show the changes recorded in the stash as a diff
- `git stash apply [<stash>]` → apply the stash on top of the current working tree state
- `git stash pop [<stash>]` → like `apply` but remove the state from the stash list

Tags

- `git tag -l` $\rightarrow$ show list of existing tags

Tags

- `git tag -l` → show list of existing tags
- `git tag [-a | -s] [-m "<message>"] <tagname> [<commit>]` → tag a commit

Tags

- `git tag -l` → show list of existing tags
- `git tag [-a | -s] [-m "<message>"] <tagname> [<commit>]` → tag a commit
- `git checkout <tagname>` → check out a working copy of the repository at a given tag

Tags

- `git tag -l` → show list of existing tags
- `git tag [-a | -s] [-m "<message>"] <tagname> [<commit>]` → tag a commit
- `git checkout <tagname>` → check out a working copy of the repository at a given tag
- `git log <tagname>..master` → get a list of changes between a given tag and the latest available version

Miscellaneous (1/2)

- `git reset [--hard] <commit>` → reset current branch to `<commit>`, keeping changes to files or not

Miscellaneous (1/2)

- `git reset [--hard] <commit>` → reset current branch to `<commit>`, keeping changes to files or not
- `git blame <file>` → show who modified each line last in `<file>`

Miscellaneous (1/2)

- `git reset [--hard] <commit>` → reset current branch to `<commit>`, keeping changes to files or not
- `git blame <file>` → show who modified each line last in `<file>`
- `git bisect [start | bad | good] [<rev>]` → use binary search to find the commit that introduced a bug

Miscellaneous (1/2)

- `git reset [--hard] <commit>` → reset current branch to `<commit>`, keeping changes to files or not
- `git blame <file>` → show who modified each line last in `<file>`
- `git bisect [start | bad | good] [<rev>]` → use binary search to find the commit that introduced a bug
- `git add -p <file>` → stage only parts of `<file>` (useful for interactive rebase)

Miscellaneous (1/2)

- `git reset [--hard] <commit>` → reset current branch to `<commit>`, keeping changes to files or not
- `git blame <file>` → show who modified each line last in `<file>`
- `git bisect [start | bad | good] [<rev>]` → use binary search to find the commit that introduced a bug
- `git add -p <file>` → stage only parts of `<file>` (useful for interactive rebase)
- `git grep` → exactly like `grep -R` but for tracked files

Miscellaneous (1/2)

- `git reset [--hard] <commit>` → reset current branch to `<commit>`, keeping changes to files or not
- `git blame <file>` → show who modified each line last in `<file>`
- `git bisect [start | bad | good] [<rev>]` → use binary search to find the commit that introduced a bug
- `git add -p <file>` → stage only parts of `<file>` (useful for interactive rebase)
- `git grep` → exactly like `grep -R` but for tracked files
- `git rev-parse --short <commit-id>` → try to abbreviate the full SHA-1 value of commit id to sorter unique value

Miscellaneous (2/2)

We mainly used *origin* as a remote until now (the default one), but you can have other tracked repositories:

- `git remote show` → show traked repositories (*remotes*)

Miscellaneous (2/2)

We mainly used *origin* as a remote until now (the default one), but you can have other tracked repositories:

- `git remote show` → show traked repositories (*remotes*)
- `git remote show <name>` → show remote branches of remote repository named `<name>`

Miscellaneous (2/2)

We mainly used *origin* as a remote until now (the default one), but you can have other tracked repositories:

- `git remote show` → show traked repositories (*remotes*)
- `git remote show <name>` → show remote branches of remote repository named `<name>`
- `git remote add <name> <url>` → add a new remote repository (instead of duplicating your local directories...)

Miscellaneous (2/2)

We mainly used *origin* as a remote until now (the default one), but you can have other tracked repositories:

- `git remote show` → show traked repositories (*remotes*)
- `git remote show <name>` → show remote branches of remote repository named `<name>`
- `git remote add <name> <url>` → add a new remote repository (instead of duplicating your local directories...)
- `git remote remove <name>` → delete the remote named `<name>`

Miscellaneous (2/2)

We mainly used *origin* as a remote until now (the default one), but you can have other tracked repositories:

- `git remote show` → show traked repositories (*remotes*)
- `git remote show <name>` → show remote branches of remote repository named `<name>`
- `git remote add <name> <url>` → add a new remote repository (instead of duplicating your local directories...)
- `git remote remove <name>` → delete the remote named `<name>`
- `git remote prune <name>` → delete all stale (already removed from the remote repository) local remote-tracking branches under `<name>`

Git rebase - interactive

- `git rebase -i <remote-current-branch>` → interactively replay commits of current branch from last in common with remote

Git rebase - interactive

- `git rebase -i <remote-current-branch>` → interactively replay commits of current branch from last in common with remote
- Git opens your favorite editor:

Git rebase - interactive

- `git rebase -i <remote-current-branch>` → interactively replay commits of current branch from last in common with remote
- Git opens your favorite editor:
  - `pick` → keep the commit (default)

Git rebase - interactive

- `git rebase -i <remote-current-branch>` → interactively replay commits of current branch from last in common with remote
- Git opens your favorite editor:
  - `pick` → keep the commit (default)
  - delete line to discard commit

Git rebase - interactive

- `git rebase -i <remote-current-branch>` → interactively replay commits of current branch from last in common with remote
- Git opens your favorite editor:
  - `pick` → keep the commit (default)
  - delete line to discard commit
  - move line up or down to reorder (be careful with dependencies)

Git rebase - interactive

- `git rebase -i <remote-current-branch>` → interactively replay commits of current branch from last in common with remote
- Git opens your favorite editor:
  - `pick` → keep the commit (default)
  - delete line to discard commit
  - move line up or down to reorder (be careful with dependencies)
  - `reword` → edit commit message

Git rebase - interactive

- `git rebase -i <remote-current-branch>` → interactively replay commits of current branch from last in common with remote
- Git opens your favorite editor:
  - `pick` → keep the commit (default)
  - delete line to discard commit
  - move line up or down to reorder (be careful with dependencies)
  - `reword` → edit commit message
  - `squash` → merge *diffs* **and** messages

Git rebase - interactive

- `git rebase -i <remote-current-branch>` → interactively replay commits of current branch from last in common with remote
- Git opens your favorite editor:
  - `pick` → keep the commit (default)
  - delete line to discard commit
  - move line up or down to reorder (be careful with dependencies)
  - `reword` → edit commit message
  - `squash` → merge *diffs* **and** messages
  - `fixup` → merge only *diffs* and keep first message

Git rebase - interactive

- `git rebase -i <remote-current-branch>` → interactively replay commits of current branch from last in common with remote
- Git opens your favorite editor:
  - `pick` → keep the commit (default)
  - delete line to discard commit
  - move line up or down to reorder (be careful with dependencies)
  - `reword` → edit commit message
  - `squash` → merge *diffs* **and** messages
  - `fixup` → merge only *diffs* and keep first message
  - `edit` → slice a commit (a bit harder than the others); requires smart use of `git add -p` and `git reset`, as shown in the demo

Git rebase - interactive

- `git rebase -i <remote-current-branch>` → interactively replay commits of current branch from last in common with remote
- Git opens your favorite editor:
  - `pick` → keep the commit (default)
  - delete line to discard commit
  - move line up or down to reorder (be careful with dependencies)
  - `reword` → edit commit message
  - `squash` → merge *diffs* **and** messages
  - `fixup` → merge only *diffs* and keep first message
  - `edit` → slice a commit (a bit harder than the others); requires smart use of `git add -p` and `git reset`, as shown in the demo
- Live demo

Conclusion

References

- *https://git-scm.com/*
- *https://git-scm.com/book/en/v2*
- *https://schacon.github.io/git/user-manual.html*
- *http://www.git-attitude.fr/2014/05/04/bien-utiliser-git-merge-et-rebase/*
- *http://free-electrons.com/docs/*
- **RTFM:** `man git-<function>` or `git help <function>`

Conclusion

- Git will quickly become your best ally for many projects

Conclusion

- Git will quickly become your best ally for many projects
- Few options were listed, read manual for more!

Conclusion

- Git will quickly become your best ally for many projects
- Few options were listed, read manual for more!
- Have faith in Git and you: there is absolutely nothing you cannot do/change/fix/revert as long as you have not pushed your work

Backup slides: Linux

For your own culture: introduction to Linux kernel development workflow

- `git commit -s` → sign your commit

For your own culture: introduction to Linux kernel development workflow

- `git commit -s` → sign your commit
- `git format-patch [-s] [<commit> | <branch>..<branch>]` → prepare patches for e-mail submission

For your own culture: introduction to Linux kernel development workflow

- `git commit -s` → sign your commit
- `git format-patch [-s] [<commit> | <branch>..<branch>]` → prepare patches for e-mail submission
- `git send-email --compose --to <people> --cc <people> <patches>` → send a collection of patches as e-mails

For your own culture: introduction to Linux kernel development workflow

- `git commit -s` → sign your commit
- `git format-patch [-s] [<commit> | <branch>..<branch>]` → prepare patches for e-mail submission
- `git send-email --compose --to <people> --cc <people> <patches>` → send a collection of patches as e-mails
- `git apply [-p <n>] [-R] <patch>` → apply a patch

For your own culture: introduction to Linux kernel development workflow

- `git commit -s` → sign your commit
- `git format-patch [-s] [<commit> | <branch>..<branch>]` → prepare patches for e-mail submission
- `git send-email --compose --to <people> --cc <people> <patches>` → send a collection of patches as e-mails
- `git apply [-p <n>] [-R] <patch>` → apply a patch
- `git am` → apply a serie of patches from a mailbox