Demystifying the Git

A ∼couple of things you need to understand to work more efficiently

Benoît "tsuna" Sigoure
<tsuna@stumbleupon.com>

June 9, 2010

StumbleUpon

Why this presentation?

"I started to understand those under-the-hood concepts [. . .] *suddenly everything made sense. I could understand the manual pages and perform all sorts of source control tasks. Everything that seemed so cryptic and obscure now was perfectly clear."*
– Charles Duan [1]

What you need to know to get started

This presentation assumes you know:

- Files, directories, how to use a terminal etc.
- Understand what version control is about.
- Bonus: know what a SHA1 is.

That's all!

Outline

Tell Git who you are

```
$ git config --global user.name "My Name"
$ git config --global user.email "my@email.example.com"
```

This will populate ~/.gitconfig with:

```
[user]
        name = My Name
        email = my@email.example.com
```

Add shiny colors

```
$ git config --global color.diff auto
$ git config --global color.status auto
$ git config --global color.branch auto
```

Unclutter your life

Various files crop up during development and clutter the output of `git status`. Don't learn to ignore them, just ignore them.

```
$ git config --global core.excludesfile ~/.gitignore
$ cat >>~/.gitignore <<EOF
*.[ao]
*.so
*.pyc
.DS_Store
\#*
.*.sw?
*~
EOF
```

Give your fingers a break

I'm lazy, and the only thing I miss from SVN is the short aliases, so let's use them with Git too!

```
$ git config --global alias.st status
$ git config --global alias.ci commit
$ git config --global alias.co checkout
$ git config --global alias.diffi diff --cached
$ git config --global alias.diffstat diff --stat
$ git config --global alias.diffw diff --ignore-all-space
$ git config --global alias.slog log --pretty=oneline
```

We'll see what those do shortly!

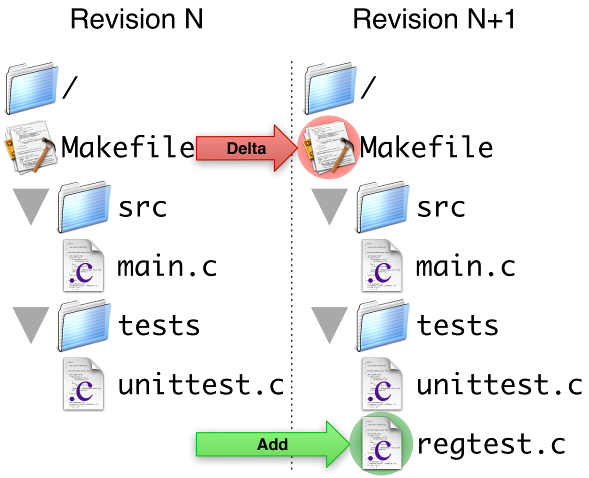
Outline

Think Git

"A lot of the good things in git come exactly from the fact that git does **not** *do things like most traditional SCM's do."*
– Linus Torvalds [4]

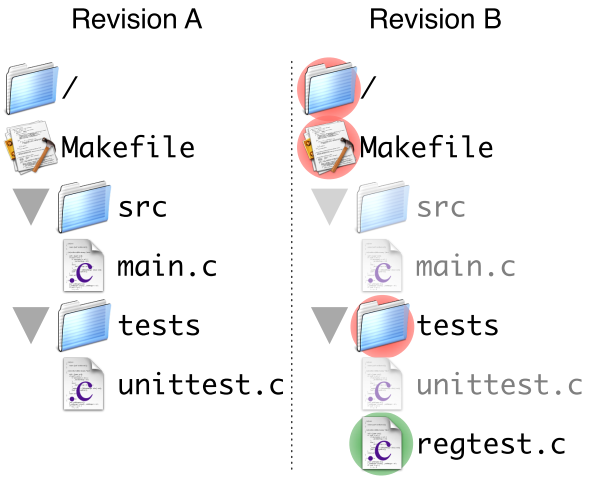
- Git doesn't track individual files, it tracks contents.
- Git has *real* branches.
- This thing called the "index".

Tracking files vs contents: the SVN model



- Each file has its own history.
- SVN stores a delta when a file is changed.

Tracking files vs contents: the Git model



- Each revision in Git has its own snapshot of the entire tree of files.
- Each element in the tree (an "object") is identified by a unique SHA1.
- Objects with the same SHA1 are shared.

Tracking files vs contents: what does it change?

Instead of storing the life of each file tracked, Git stores separately *content*, *history* and *structure*. [3]

- Content is stored in *blob* objects.
  Blobs have no name.
- Structure is stored in *tree* objects.
  Trees organize blobs in a named hierarchy.
- History is stored in *commit* objects.
  Commits keep trees in a chronological order.

Tracking files vs contents: what does it change?

Instead of storing the life of each file tracked, Git stores separately *content*, *history* and *structure*. [3]

- Content is stored in *blob* objects.
  Blobs have no name.
- Structure is stored in *tree* objects.
  Trees organize blobs in a named hierarchy.
- History is stored in *commit* objects.
  Commits keep trees in a chronological order.

Main consequence:

If two files have the same contents, they're the same file.
The trees referencing them may do so under a different name.

Tracking files vs contents: what does it change?

If two files have the same contents, they're the same file.
The trees referencing them may do so under a different name.

$\Rightarrow$ Git doesn't need to be told when a file is renamed.
Merging different branches in which files have moved around Just Works!

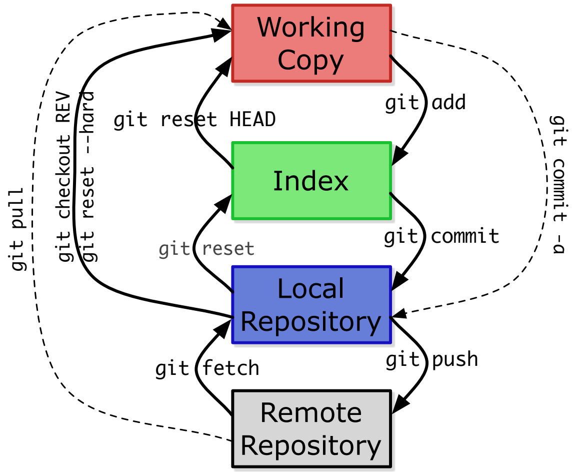
Outline

What the heck is the “index”?

- Most SCMs have just a working copy and a repository.
- Whatever you do in the working copy will be in the next commit.
- Often you can commit a subset of the files at a time.

What the heck is the “index”?

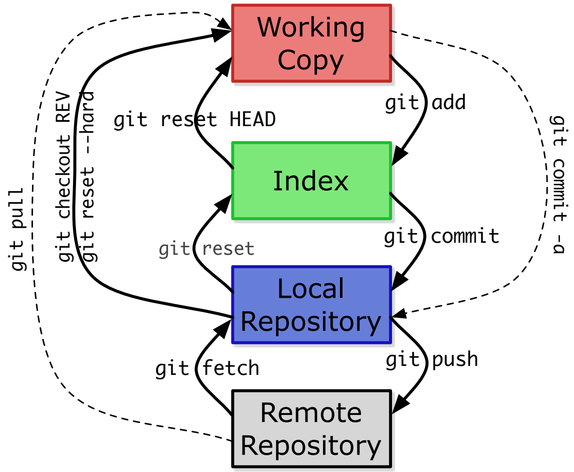
- Git has a third thing in between: the index
- It’s a staging area. Whatever is in it will be in the next commit.
- You have a very fine grained control over the index.



StumbleUpon

What the heck is the "index"?

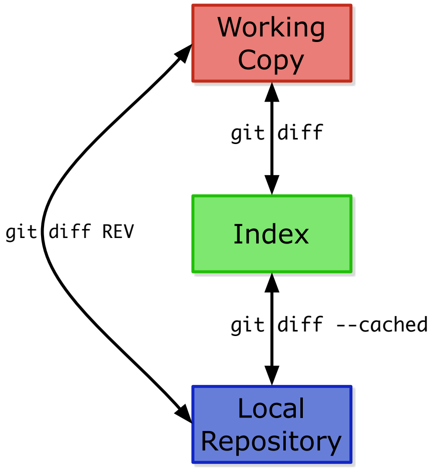
- Where is it? In a file under the `.git/` directory.
- What does it change? It's one of the killer features of Git!
- You can chose exactly what bits of changes go into the index.



StumbleUpon

What the heck is the "index"?

- `git diff` can show you the differences between each layer.
- By default, only the differences with the index are shown.
- To ignore the index, specify a revision to compare with.



The index in practice...

```
$ git diff
diff --git a/quotes.txt b/quotes.txt
index f16f045..200de06 100644
--- a/quotes.txt
+++ b/quotes.txt
@@ -1,7 +1,10 @@
 Humans are not proud of their ancestors, and rarely invite
 them round to dinner.
+Time is an illusion. Lunchtime doubly so.
 I may not have gone where I intended to go, but I think I
 have ended up where I needed to be.
 In the beginning the Universe was created. This has made a
 lot of people very angry and has been widely regarded as a
 bad move.
+It is a mistake to think you can solve any major problems
+just with potatoes.
```

The index in practice...

```
$ git add -i
           staged     unstaged path
  1:    unchanged        +3/-0 quotes.txt

*** Commands ***
  1: [s]tatus   2: [u]pdate   3: [r]evert   4: [a]dd untracked
  5: [p]atch    6: [d]iff     7: [q]uit     8: [h]elp
What now>
```

The index in practice...

```
$ git add -i
           staged     unstaged path
  1:    unchanged        +3/-0 quotes.txt

*** Commands ***
  1: [s]tatus   2: [u]pdate   3: [r]evert   4: [a]dd untracked
  5: [p]atch    6: [d]iff     7: [q]uit     8: [h]elp
What now> p
           staged     unstaged path
  1:    unchanged        +3/-0 [q]uotes.txt
Patch update>>
```

The index in practice...

```
$ git add -i
           staged     unstaged path
  1:    unchanged        +3/-0 quotes.txt

*** Commands ***
  1: [s]tatus   2: [u]pdate   3: [r]evert   4: [a]dd untracked
  5: [p]atch    6: [d]iff     7: [q]uit     8: [h]elp
What now> p
           staged     unstaged path
  1:    unchanged        +3/-0 [q]uotes.txt
Patch update>> 1
           staged     unstaged path
* 1:    unchanged        +3/-0 [q]uotes.txt
Patch update>> (hit enter)
```

The index in practice...

```
diff --git a/quotes.txt b/quotes.txt
index f16f045..200de06 100644
--- a/quotes.txt
+++ b/quotes.txt
@@ -1,7 +1,10 @@
 Humans are not proud of their ancestors, and rarely invite
 them round to dinner.
+Time is an illusion. Lunchtime doubly so.
 I may not have gone where I intended to go, but I think I
 have ended up where I needed to be.
 In the beginning the Universe was created. This has made a
 lot of people very angry and has been widely regarded as a
 bad move.
+It is a mistake to think you can solve any major problems
+just with potatoes.
Stage this hunk [y,n,q,a,d,/,s,e,?]? s  (split)
```

The index in practice...

```
Split into 2 hunks.
@@ -1,7 +1,8 @@
 Humans are not proud of their ancestors, and rarely invite
 them round to dinner.
+Time is an illusion. Lunchtime doubly so.
 I may not have gone where I intended to go, but I think I
 have ended up where I needed to be.
 In the beginning the Universe was created. This has made a
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? y  (yes, add it)
```

The index in practice...

```
Split into 2 hunks.
@@ -1,7 +1,8 @@
 Humans are not proud of their ancestors, and rarely invite
 them round to dinner.
+Time is an illusion. Lunchtime doubly so.
 I may not have gone where I intended to go, but I think I
 have ended up where I needed to be.
 In the beginning the Universe was created. This has made a
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? y
@@ -5,3 +6,5 @@
 In the beginning the Universe was created. This has made a
 lot of people very angry and has been widely regarded as a
 bad move.
+It is a mistake to think you can solve any major problems
+just with potatoes.
Stage this hunk [y,n,q,a,d,/,K,g,e,?]? n  (no)
```

The index in practice...

```
$ git st
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   quotes.txt
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in ...)
#
#       modified:   quotes.txt
```

The index in practice...

```
$ git diff
diff --git a/quotes.txt b/quotes.txt
index 93fc987..200de06 100644
--- a/quotes.txt
+++ b/quotes.txt
@@ -6,3 +6,5 @@ have ended up where I needed to be.
 In the beginning the Universe was created. This has made a
 lot of people very angry and has been widely regarded as a
 bad move.
+It is a mistake to think you can solve any major problems
+just with potatoes.
```

The index in practice...

```
$ git diffi  (my alias for diff --cached)
diff --git a/quotes.txt b/quotes.txt
index f16f045..93fc987 100644
--- a/quotes.txt
+++ b/quotes.txt
@@ -1,5 +1,6 @@
 Humans are not proud of their ancestors, and rarely invite
 them round to dinner.
+Time is an illusion. Lunchtime doubly so.
 I may not have gone where I intended to go, but I think I
 have ended up where I needed to be.
 In the beginning the Universe was created. This has made a
```

The index in practice...

```
$ git diffi
diff --git a/quotes.txt b/quotes.txt
index f16f045..93fc987 100644
--- a/quotes.txt
+++ b/quotes.txt
@@ -1,5 +1,6 @@
 Humans are not proud of their ancestors, and rarely invite
 them round to dinner.
+Time is an illusion. Lunchtime doubly so.
 I may not have gone where I intended to go, but I think I
 have ended up where I needed to be.
 In the beginning the Universe was created. This has made a
$ git commit -m 'Add a quote about time'
[master 34a68e5] Add a quote about time
 1 files changed, 1 insertions(+), 0 deletions(-)
```

The index in practice...

```
$ git commit -a -m 'Add another quote'
[master c04c938] Add another quote
 1 files changed, 2 insertions(+), 0 deletions(-)
$ git slog
c04c938df8c1dcfb7d55938b2316f59f42ab29ee Add another quote
34a68e548f8755324bf79bb02e7c9c8ce0f9acd5 Add a quote about time
[...]
```

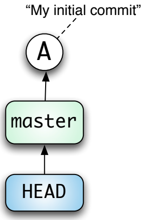
Outline

Overview of what you'll frequently need

- New branch: `git co -b mybranch origin/master`
- What's new in my branch? `git log ...origin/master` (3 dots, will explain later)
- Sync up with others: `git pull --rebase`

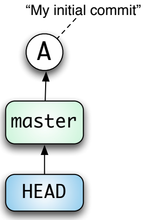
Create a new repository

```
$ git init   # Create a brand new repository
<create directories and files>
$ git add .  # Add all files in the index
$ git commit -m 'My initial commit'
```



Create a new repository

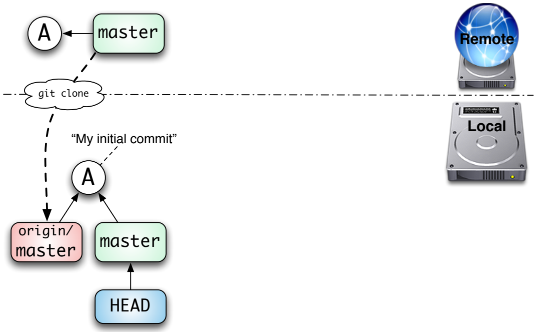
```
$ git init    # Create a brand new repository
<create directories and files>
$ git add .   # Add all files in the index
$ git commit -m 'My initial commit'
```



- Commit `A` doesn't have a parent.
- Commit `A` is reachable through the branch `master`.
- `HEAD` is maintained by Git to always point to the head of the current branch.

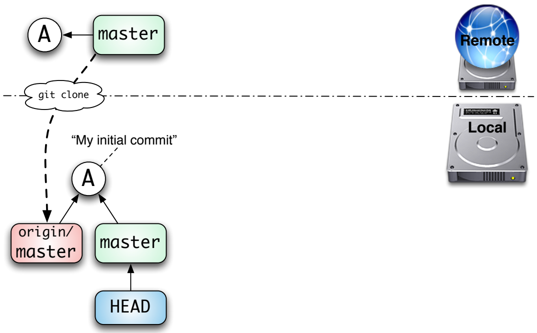
Clone an existing repository

```
$ git clone url://to/repository
Initialized empty Git repository in .../repository/.git/
[...]
```



Clone an existing repository

```
$ git clone url://to/repository
Initialized empty Git repository in .../repository/.git/
[...]
```



The remote repository is copied locally.

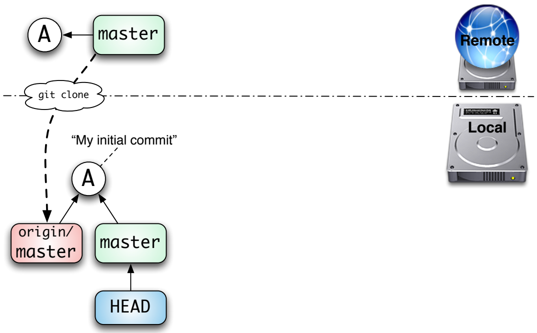
Every remote branch appears locally under its own namespace called `origin`.

`origin` is a "remote".

You can have as many remotes as you want.

Clone an existing repository

```
$ git clone url://to/repository
Initialized empty Git repository in .../repository/.git/
[...]
```



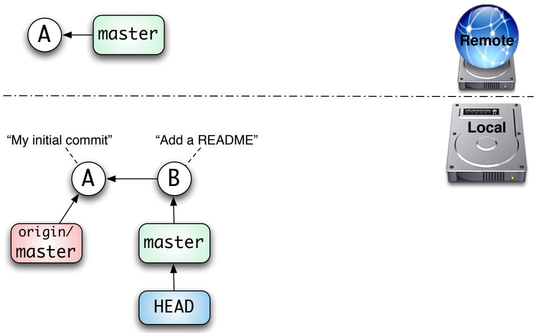
Git automatically created a local branch called `master`.

Your `master` is in the same state as its remote counterpart, `origin/master`.

```
$ git branch -a
* master
  remotes/origin/master
```

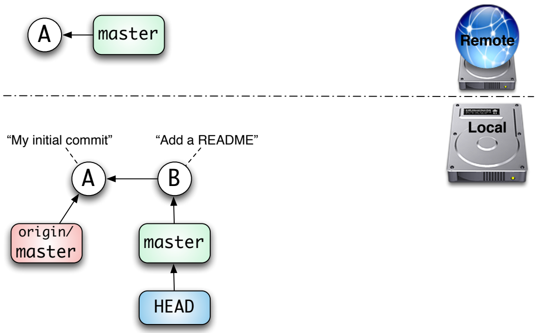
Let's create some commits to pretend we're working

```
$ echo 'Hello World!' >README && git add README
$ git commit -m 'Add a README'
[master B] Add a README
```



Let’s create some commits to pretend we’re working

```
$ echo ’Hello World!’ >README && git add README
$ git commit -m ’Add a README’
[master B] Add a README
```

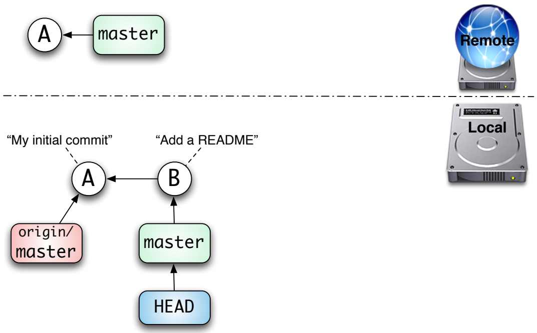


A is the parent commit of B.

Commit B only exists in our own private repository.

Let's create some commits to pretend we're working

```
$ echo 'Hello World!' >README && git add README
$ git commit -m 'Add a README'
[master B] Add a README
```

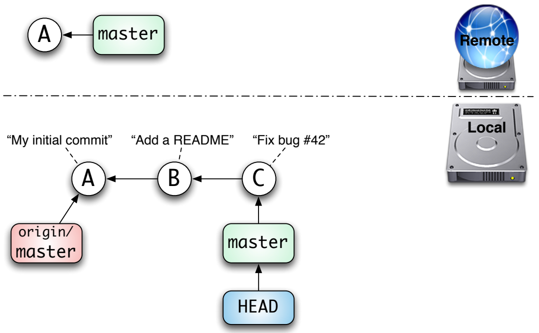


Our `master` branch is now 1 commit ahead of `origin`'s.

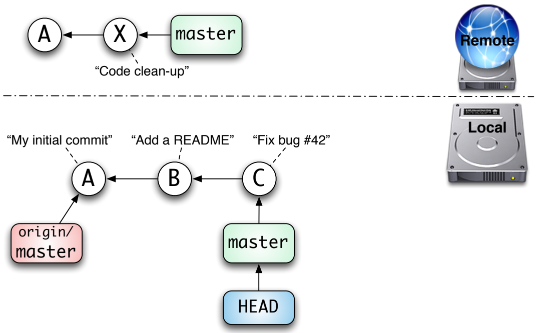
```
$ git st
# On branch master
# Your branch is ahead of
'origin/master' by 1
commit.
# nothing to commit
(working directory clean)
```

Let's work some more

```
<hack hack hack>
$ git commit -m 'Fix bug #42'
[master C] Fix bug #42
```



In the mean time. . .



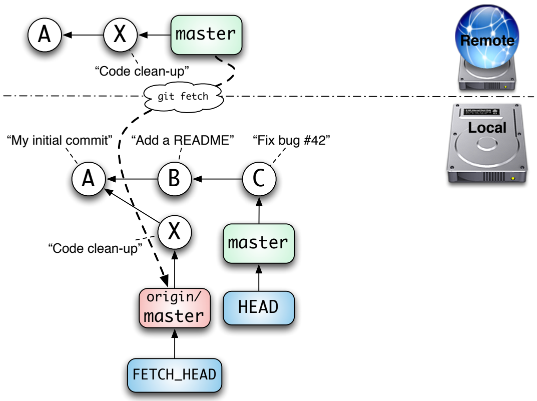
Someone else pushed a commit `X` to the remote repository.

This change isn't visible to us yet as it hasn't made it to our local repository.

StumbleUpon

Get the remote changes

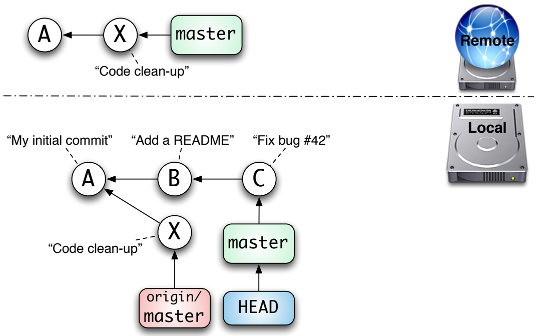
```
$ git fetch
[...]
   A..X  master      -> origin/master
```



We updated our view of the remote `origin`.

`FETCH_HEAD` is created to provide references to all the remote heads we just retrieved.

Integrate the remote changes



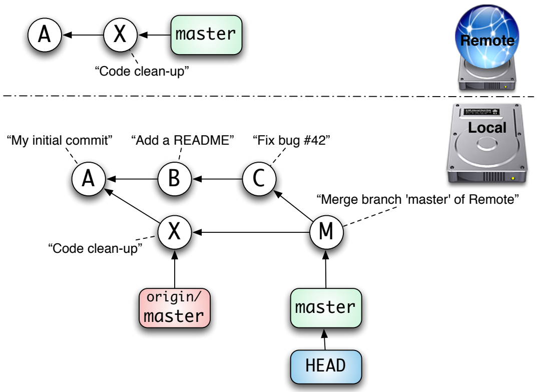
There are two ways to integrate your changes with their changes.

- Join both lines of development together
  `git merge`
- Keep a single, linear line of development
  `git rebase`

StumbleUpon

Integrate the remote changes with a merge

```
$ git merge origin/master
Merge made by recursive.
 2 files changed, 13 insertions(+), 42 deletions(-)
```



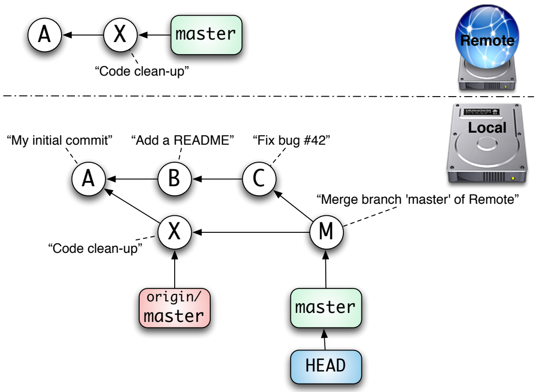
M is a merge commit: it has 2 parents (C and X).

Our history is non-linear and reflects 2 parallel lines of development.

If there are conflicting changes, the conflict resolution is recorded in M.

Integrate the remote changes with a merge

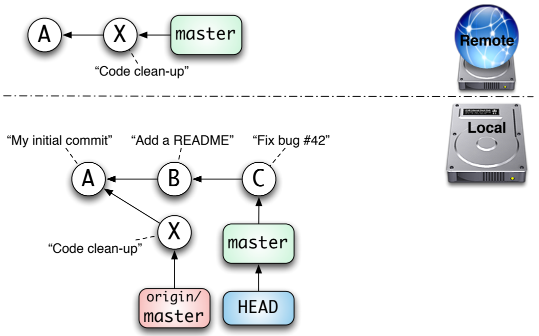
```
$ git merge origin/master
Merge made by recursive.
 2 files changed, 13 insertions(+), 42 deletions(-)
```



Several people + frequent integrations
= many merges
= history hard to read

Because
`pull = fetch + merge`
most people do it

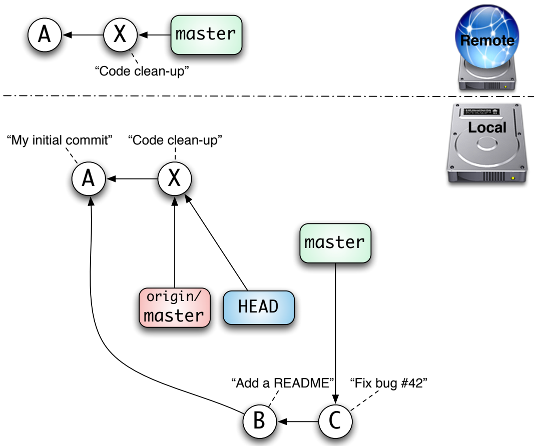
Integrate the remote changes with rebase



StumbleUpon

Integrate the remote changes with rebase

```
$ git rebase origin/master
First, rewinding head to replay your work on top of it...
```

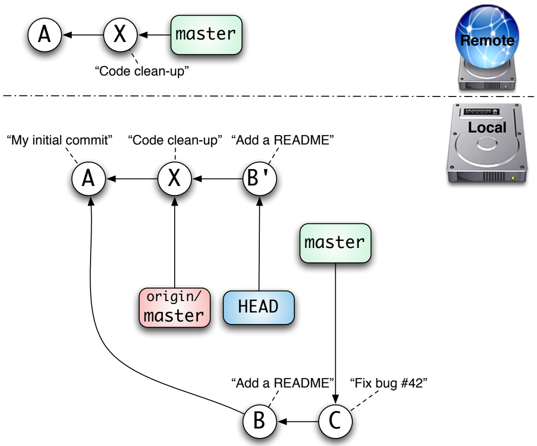


Step 1: Catch up with the remote

Note that HEAD doesn't point to any branch, it's "detached".

Integrate the remote changes with rebase

```
$ git rebase origin/master
First, rewinding head to replay your work on top of it...
Applying: Add a README
```

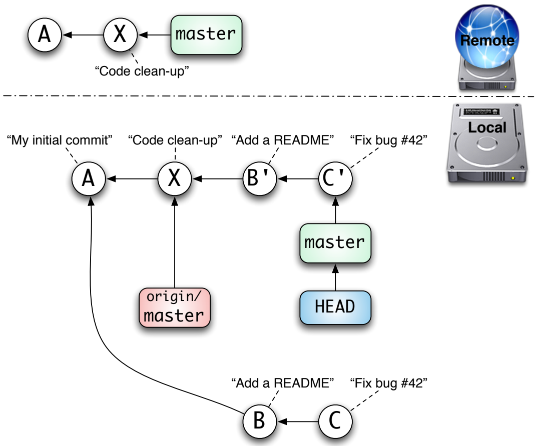


Step 2: Re-apply each of your commits

Integrate the remote changes with rebase

```
$ git rebase origin/master

Applying: Fix bug #42
```

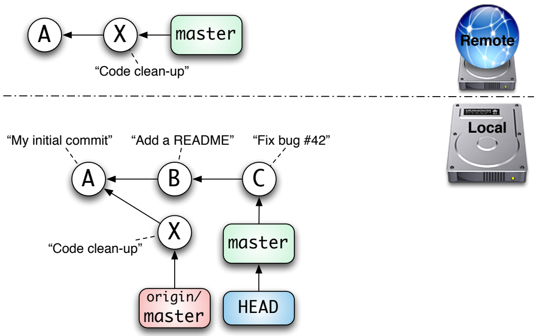


Git *rewrote* the history. Now it appears as if `B'` and `C'` were created after `X`.

The original `B` and `C` are not lost, they're just not reachable through any branch (garbage).

You can also use `git pull --rebase` for `fetch + rebase`.

When to use merge vs rebase



Both merge and rebase find the common ancestor to know what to merge / rebase.

`A` is the common ancestor of `origin/master` and `master`.

When to use merge vs rebase

If history has already been published, don't rewrite it!

$\Rightarrow$ Don't rebase commits you already pushed.
$\Rightarrow$ Don't rebase others' commits.

Outline

When things don't work out as expected

DON'T PANIC

You can't break a Git repository[*][†]

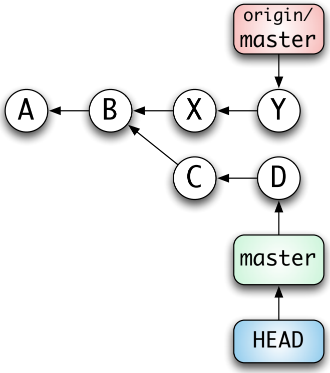
- Deleting and re-creating your repository is never the right solution to your problem and may cause permanent data loss.
- You can always "go back" to a previously known good state.
- Git records every action you take in the `reflog`. You can use it to undo every step you take.
- Git will never lose[†], modify or corrupt existing commits.

[*]Fiddling with things under `.git` may void your warranty.

[†]Running `git gc` may prevent you from undoing a mistake.

After a bad rebase

By far the most frequent problem I see when people start using rebase.

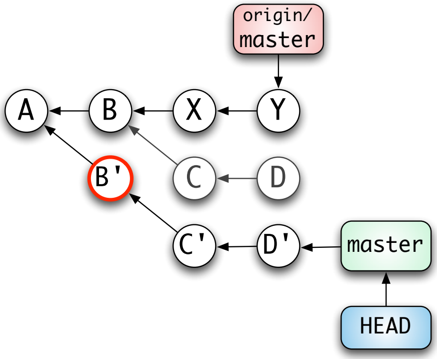


"I'm seeing weird conflicts on files I haven't touched when doing a rebase."

Most of the time it's due to a bad rebase done earlier, sometimes days before (so you don't realize it's what's causing trouble now).

After a bad rebase

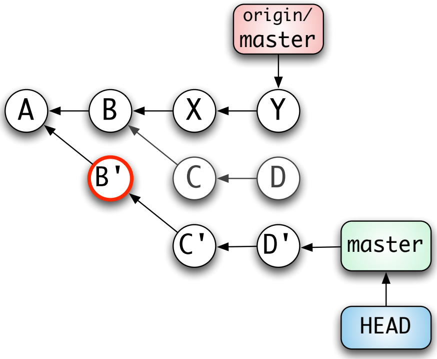
Typical case: `B` was rebased even though it was already pushed.



Later Git will find that the common ancestor between `master` and `origin/master` is `A` instead of `B`, and it will be confused by the duplicated commit `B'`.

After a bad rebase

Typical case: `B` was rebased even though it was already pushed.

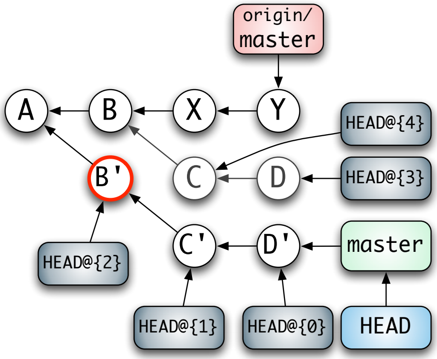


Possible fixes:

- You're still in the middle of the rebase:
  `git rebase --abort`
- Use the `reflog` to go back before the rebase.
- Rebuild your branch with `cherry-pick`.

Go back to a known good state with the `reflog`

You can undo the rebase by identifying which revision you were at before.



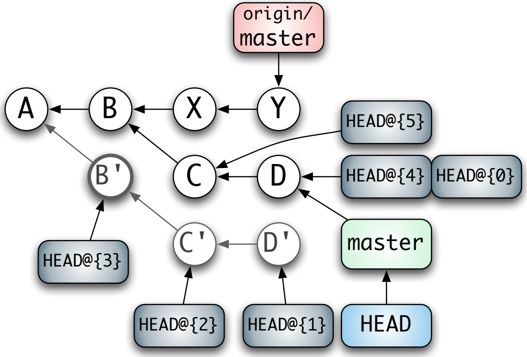
Identify the revision you want to go back to:

```
$ git reflog
D’ HEAD@{0} rebase:  ..
C’ HEAD@{1} rebase:  ..
B’ HEAD@{2} rebase:  ..
D HEAD@{3} commit:  ..
C HEAD@{4} commit:  ..
```

Before the rebase we were at `HEAD@{3}`

Go back to a known good state with the `reflog`

You can undo the rebase by identifying which revision you were at before.



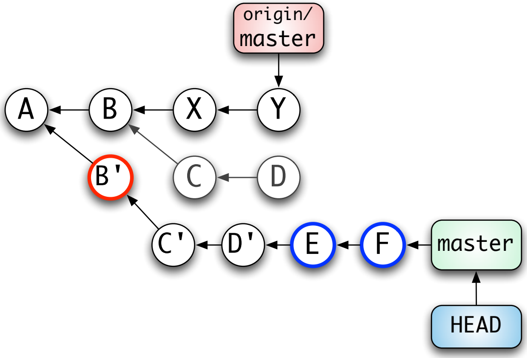
So let's go back to `HEAD@{3}`

```
$ git reset --hard HEAD@{3}
```

`--hard` resets everything: all files (make sure you don't have any pending change!), the index, and the head (`master` and `HEAD`)

Go back to a known good state with `cherry-pick`

Sometimes you did some work after the bad rebase.

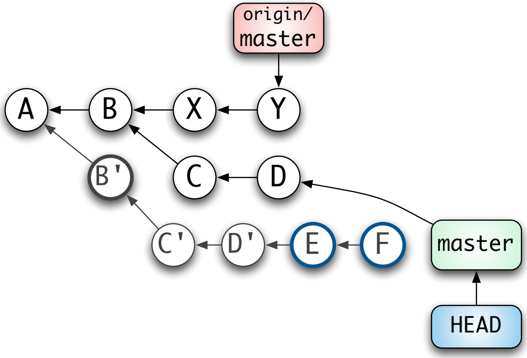


Use the same method as previously to undo the bad rebase.

```
$ git reset --hard HEAD@{3}
```

Go back to a known good state with `cherry-pick`

Sometimes you did some work after the bad rebase.



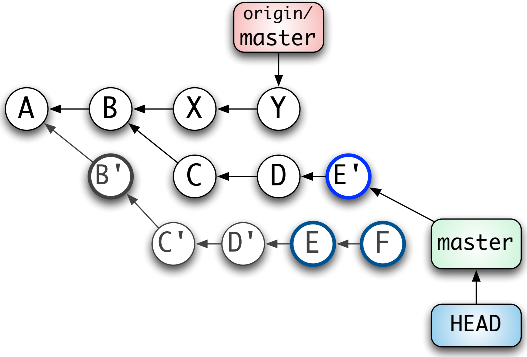
Oops, now we're missing `E` and `F`!

You can "cherry pick" them back into your branch.

```
git cherry-pick E
```

Go back to a known good state with cherry-pick

Sometimes you did some work after the bad rebase.

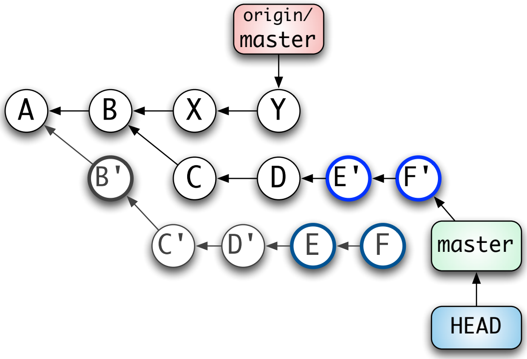


One more...

```
git cherry-pick F
```

Go back to a known good state with `cherry-pick`

Sometimes you did some work after the bad rebase.

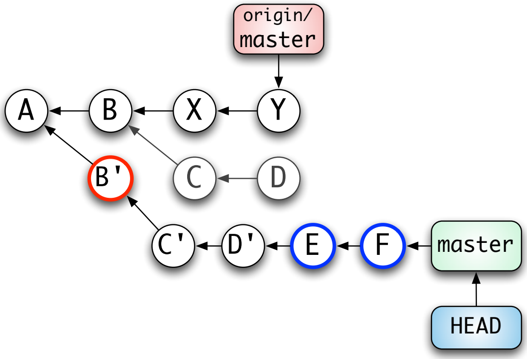


What we did manually is very much like what `git rebase` does.

Actually... You can use `git rebase` to make the process simpler!

Go back to a known good state with rebase

In interactive mode, you can tell `git rebase` what you want.



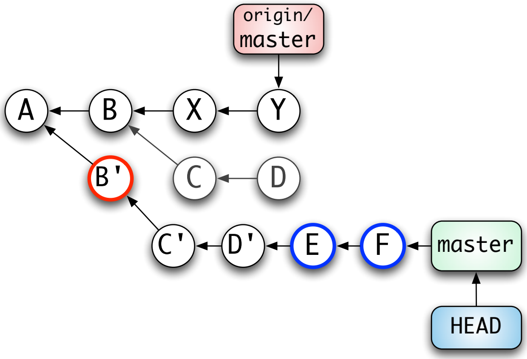
```
$ git rebase -i
origin/master
```

Your editor will open:

```
pick B' ..
pick C' ..
pick D' ..
pick E ..
pick F ..
```

Go back to a known good state with rebase

In interactive mode, you can tell `git rebase` what you want.



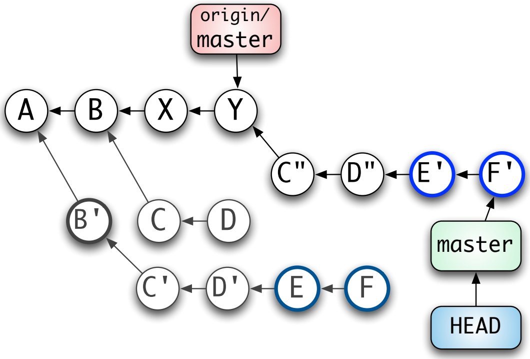
Remove the commits that aren't yours.

In this case, just delete the first line:

```
pick C' ..
pick D' ..
pick E ..
pick F ..
```

Go back to a known good state with rebase

In interactive mode, you can tell `git rebase` what you want.



`git rebase` will cherry-pick the commits you specified and put them on top of `origin/master`.

Outline

When creating a new branch...

Make sure you use

```
git co -b mybranch origin/master
```

This way Git will record (in `.git/config`) the fact that `mybranch` tracks `origin/master`.

This way `git pull` won't complain:

```
You asked me to pull without telling me which branch you
want to rebase against, and 'branch.mybranch.merge' in
your configuration file does not tell me, either. Please
specify which branch you want to use on the command line and
try again (e.g. 'git pull <repository> <refspec>').
See git-pull(1) for details.
[...]
```

StumbleUpon

Use the stash

```
$ git st
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in ...)
#
#       modified:   quotes.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

You're in the middle of something, someone interrupts you:

> *"OMG, we just pushed this critical bug in your code, please fix it ASAP!"*

No, problem, `git stash` to the rescue!

Use the stash

```
$ git st
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in ...)
#
#       modified:   quotes.txt
no changes added to commit (use "git add" and/or "git commit -a")
$ git stash
Saved working directory and index state WIP on master: 34a68e5 Add
HEAD is now at 34a68e5 Add a quote about time
```

Use the stash

```
$ git st
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in ...)
#
#       modified:   quotes.txt
no changes added to commit (use "git add" and/or "git commit -a")
$ git stash
Saved working directory and index state WIP on master: 34a68e5 Add
HEAD is now at 34a68e5 Add a quote about time
$ git st
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
nothing to commit (working directory clean)
```

Use the stash

- Now that everything is out of the way, you can fix that bug.
- Commit the fix.
- Push it out.
- `git stash pop`
- Resume what you were doing.

If you use `git stash` multiple times, changes get stacked up.
`git stash list` shows the stack of stashes.

Useful lil things

- `git grep` *regexp*: Recursive grep in the current directory.
- `git show` *revision*: Look at a given commit.
- `git log --stat`: Look at the history and see which files are affected by each commit.
- Forgot something in your last commit? And you haven't pushed yet? Fix it and `git commit -a --amend` to rewrite the last commit.
- Want to return in the state *just before* the last `git commit`?
  `git reset HEAD~1`
- Want to get rid of all your pending changes?
  `git reset --hard` — but think twice as there's no going back with this one!
- Git performs a **lot** better on a local filesystem than on NFS.

Outline

StumbleUpon

TBD

TBD

Outline

TBD

TBD

Outline

Conclusion

Git has a steep learning curve! But once you get past the initial learning period, you save a lot of time with it.

- The index is your friend.
- Don't be afraid of rebase.
- When in trouble, call the `reflog` to the rescue.
- TBD

Time for Q'n'A!

Don't forget that the guy currently behind Git is Junio Hamano!
He does all the hard work and deserves all the credit!

StumbleUpon

Outline

He said it

- “I’m an egotistical bastard, and I name all my projects after myself. First Linux, now git.” [2]
- “For the first 10 years of kernel maintenance, we literally used tarballs and patches, which is a much superior source control management system than CVS is”. [5]
- “I like colorized diffs, but let’s face it, those particular color choices will make most people decide to pick out their eyes with a fondue fork. And that’s not good. Digging in your eye-sockets with a fondue fork is strictly considered to be bad for your health, and seven out of nine optometrists are dead set against the practice. So in order to avoid a lot of blind git users, please apply this patch.”
- “I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships.”

He said it

- “Nobody actually creates perfect code the first time around, except me. But there’s only one of me.” [5]
- “If you have ever done any security work and it did not involve the concept of ”network of trust”, it wasn’t security work, it was masturbation. I don’t know what you were doing but trust me, it’s the only way you can do security, and it’s the only way you can do development. The way I work, I don’t trust everybody. In fact I am a very cynical and untrusting person. I think most of you are completely incompetent” [5]

Trivia: Did you know that Git became self-hosting on its 4th day of development, just 1 day after being announced?

Bibliography I

[1] Charles Duan.
Understanding git conceptually, April 2010.
URL http://www.eecs.harvard.edu/~cduan/technical/git/

[2] Robert McMillan and Linus Torvalds.
After controversy, torvalds begins work on "git".
PC World Business Australia, April 2005.
URL http://www.pcworld.idg.com.au/article/129776/after_controversy_torvalds_begins_work_git_/

[3] Jakub Narebski et al.
Git Wiki, January 2010.
URL https://git.wiki.kernel.org/index.php/Git

[4] Linus Torvalds.
Re: Vcs comparison table.
Git Mailing List, October 2006.
URL http://marc.info/?l=git&m=116129092117475

[5] Linus Torvalds.
Tech talk: Linus torvalds on git.
@Google Tech Talk, May 2007.
URL http://www.youtube.com/watch?v=4XpnKHJAok8