Git

Mastering the Beast

Florian Vaussard

HEIG-VD REDS

21st September 2016

Outline

Outline

What is Git?

History

Git is born in 2005 due to licensing issues between Linux and BitKeeper

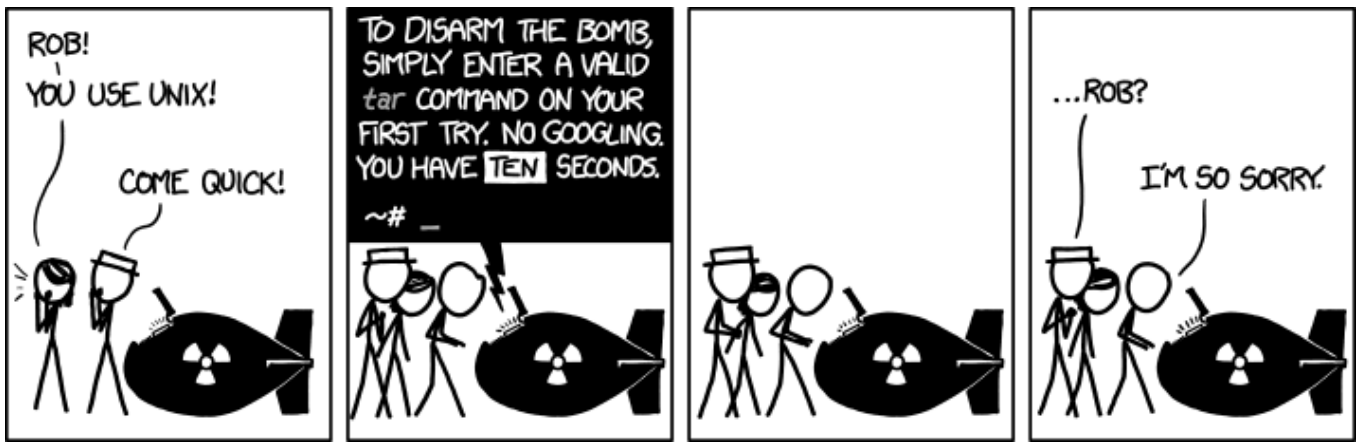
Some cool features ($\neq$ svn)

- Git is fast
- Easy branching and merging
- Distributed
- Work offline, experiment locally
- No server even needed

This Presentation

In this presentation

- Learn **what you can do** with Git [and optionally **how**]
- Biased towards Linux and CLI
  - Install Git CLI on Windows: `https://git-scm.com/download/win`
- Biased towards software development
  - Git is less useful with binary objects



Outline

Configuration

Mandatory

```
$ git config --global user.name "Florian Vaussard"
$ git config --global user.email "florian.vaussard@heig-vd.ch"
```

See also `~/.gitconfig`

Getting Help

Git has really good man pages

```
$ apt-get install git-man
```

Help for "`git cmd`"

```
$ git help cmd
```

or

```
$ man git-cmd
```

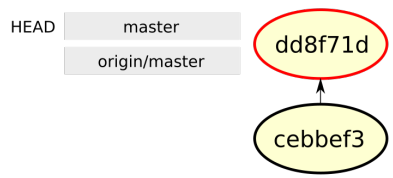
Other resources: `https://git-scm.com`, Google, Stackoverflow

Basic Workflow

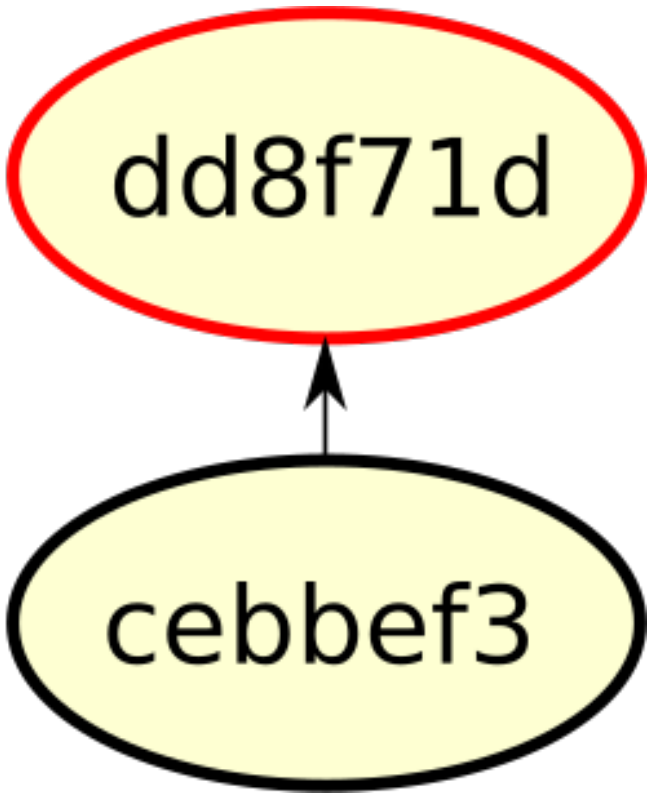
```
$ git clone git@redsmine.heig-vd.ch:project
$ cd project
```

Basic Workflow

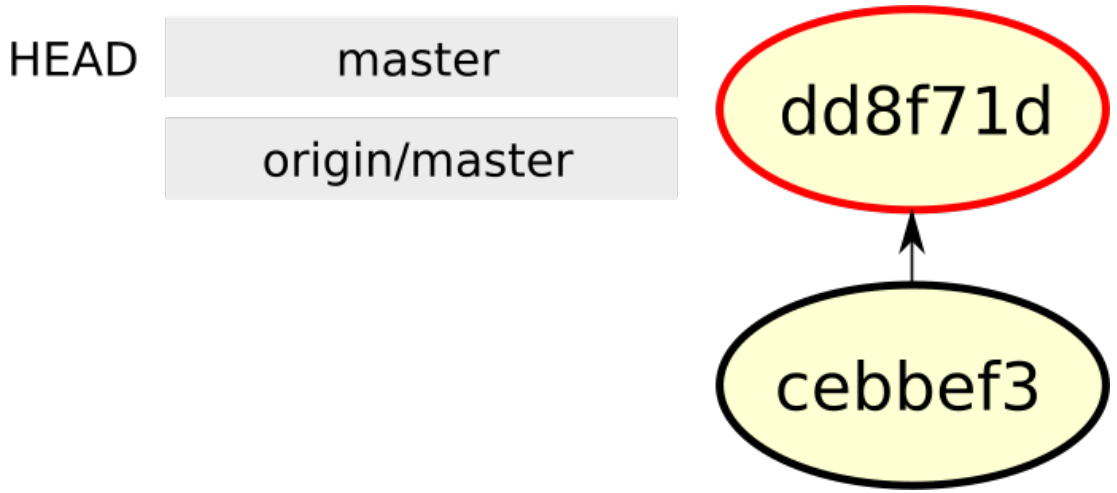
```
$ git clone git@redsmine.heig-vd.ch:project
$ cd project
```



Basic Workflow

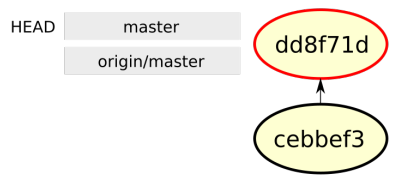


Basic Workflow



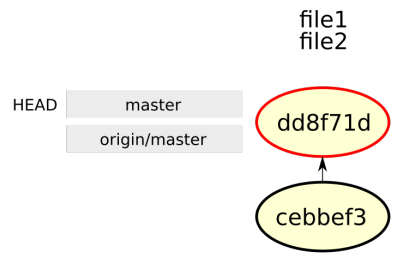
Basic Workflow

Edit file1, file2...



Basic Workflow

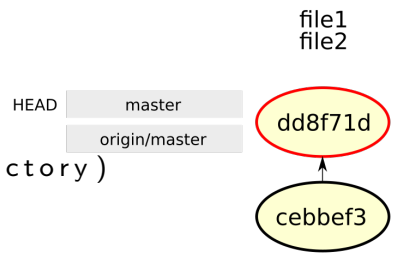
```
$ git status
```



Basic Workflow

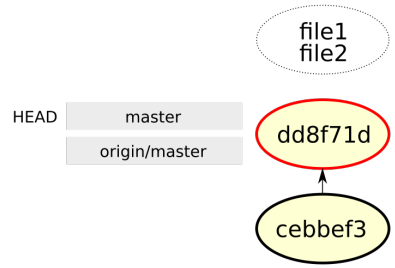
```
$ git status
```

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified: file1

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        file2
```



Basic Workflow

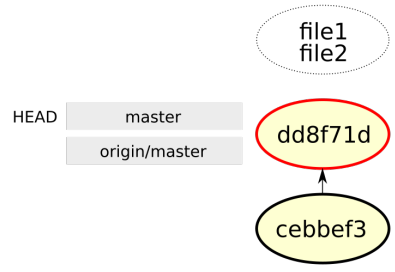
```
$ git status

$ git add file1 file2
$ git status
```



Basic Workflow

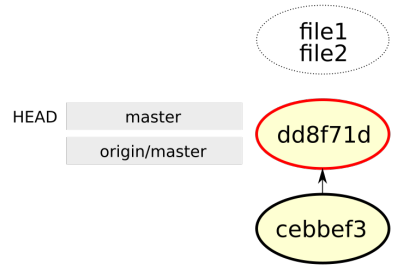
```
$ git status

$ git add file1 file2
$ git status
```

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified: file1
        new file: file2
```



Basic Workflow

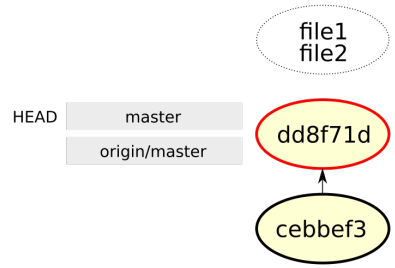
```
$ git status

$ git add file1 file2
$ git status

$ git diff --cached
```



Basic Workflow

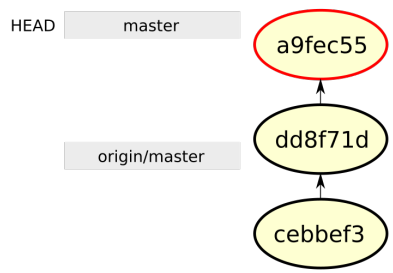
```
$ git status

$ git add file1 file2
$ git status

$ git diff --cached
```

```
$ git diff --cached
diff --git a/file1 b/file1
index 3b18e51..557db03 100644
--- a/file1
+++ b/file1
@@ -1 +1 @@
-hello world
+Hello World
diff --git a/file2 b/file2
new file mode 100644
index 0000000..d77231c
--- /dev/null
+++ b/file2
@@ -0,0 +1 @@
+FILE2
```



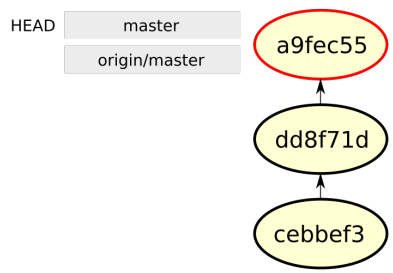
Basic Workflow

```
$ git status

$ git add file1 file2
$ git status

$ git diff --cached

$ git commit
```



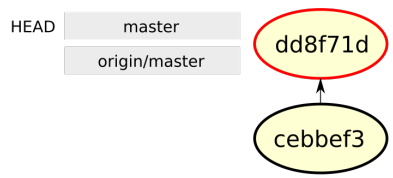
Basic Workflow

```
$ git status

$ git add file1 file2
$ git status

$ git diff --cached

$ git commit

$ git push
```



Basic Workflow

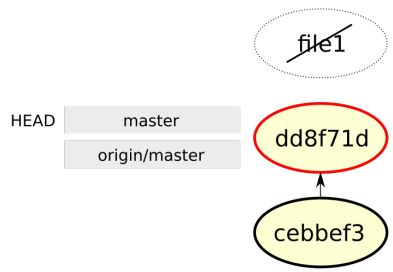
Removing a tracked file



Basic Workflow

Removing a tracked file

```
$ git rm file1
```

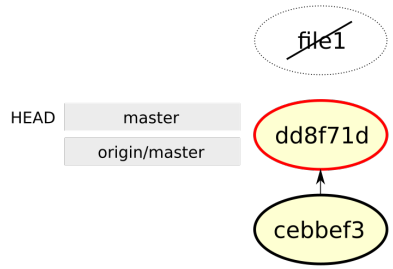


Basic Workflow

Removing a tracked file

```
$ git rm file1
$ git status
```

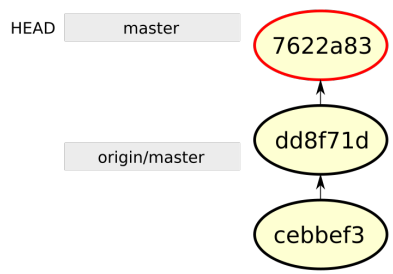
```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:  file1
```



Basic Workflow

Removing a tracked file

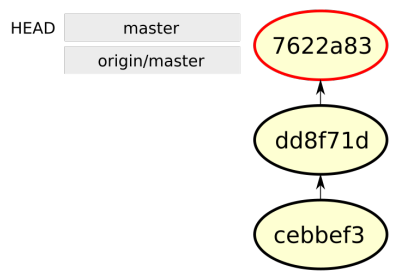
```
$ git rm file1
$ git status
$ git commit -m "Removing useless file"
```



Basic Workflow

Removing a tracked file

```
$ git rm file1
$ git status
$ git commit -m "Removing useless file"
$ git push
```

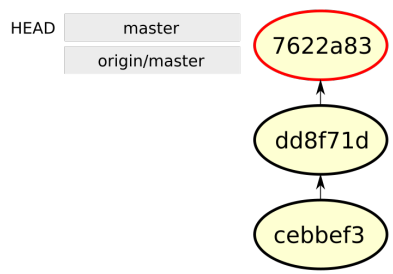


Basic Workflow

Removing a tracked file

```
$ git rm file1
$ git status
$ git commit -m "Removing useless file"
$ git push
```

Do not be afraid, files are never lost!
(unless you try very hard)



Basic Workflow

Moving/renaming a file

Basic Workflow

Moving/renaming a file

```
$ git mv /file/to/move /new/location
```

Basic Workflow

Moving/renaming a file

```
$ git mv /file/to/move /new/location

$ git status
$ git commit
$ git push
```

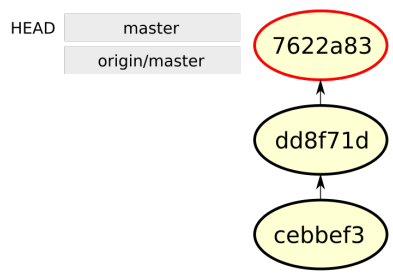
Basic Workflow

Moving/renaming a file

```
$ git mv /file/to/move /new/location

$ git status
$ git commit
$ git push
```

Do not move the file by hand!

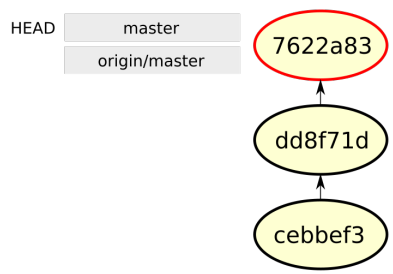
Inspecting the History



Inspecting the History

```
$ git log
```

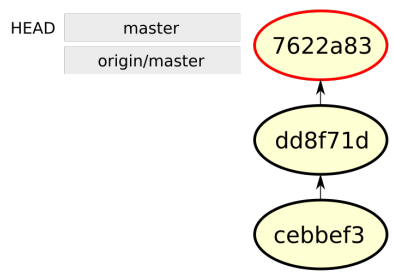
```
$ git log
commit 7622a834a536e3de61e6aee0641cea52884c8961
Author: Florian Vaussard <florian.vaussard@heig-vd.ch>
Date:   Fri Mar 11 14:20:19 2016 +0100

    Removing useless file

commit dd8f71d70f6b3710136fe9e393c8cecdf3839431
Author: Florian Vaussard <florian.vaussard@heig-vd.ch>
Date:   Fri Mar 11 13:52:28 2016 +0100

    Add readme

commit cebbef3aca0abd8be152513c78d25cdbfbdddf78
Author: Florian Vaussard <florian.vaussard@heig-vd.ch>
Date:   Fri Mar 11 13:52:03 2016 +0100

    Initial commit
```



Inspecting the History

```
$ git log
$ git log --oneline
```

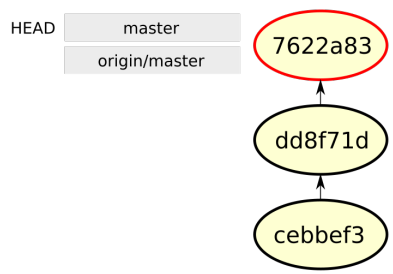
```
$ git log --oneline
7622a83 Removing useless file
dd8f71d Add readme
cebbef3 Initial commit
```



Inspecting the History

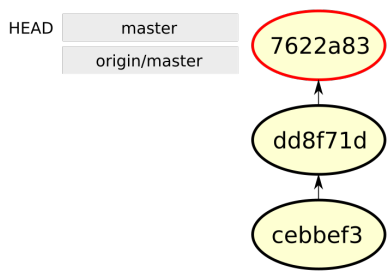
```
$ git log
$ git log --oneline
$ git log --oneline /path/to/file
```

```
$ git log --oneline file1
7622a83 Removing useless file
cebbef3 Initial commit
```



Inspecting the History

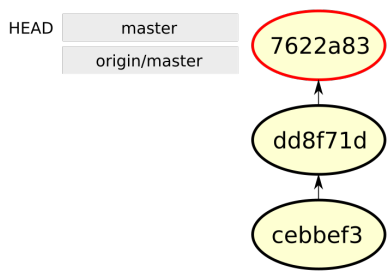
```
$ git log
$ git log --oneline
$ git log --oneline /path/to/file
$ git log SHA1..SHA1
```



Inspecting the History

```
$ git log
$ git log --oneline
$ git log --oneline /path/to/file
$ git log SHA1..SHA1
```

Can be useful: **qgit, gitk**



Inspecting the History

```
$ git show SHA1
```

Inspecting the History

```
$ git show SHA1
```

```
$ git show 7622a83
commit 7622a834a536e3de61e6aee0641cea52884c8961
Author: Florian Vaussard <florian.vaussard@heig-vd.ch>
Date:   Fri Mar 11 14:20:19 2016 +0100

    Removing useless file

diff --git a/file1 b/file1
index 3b18e51..557db03 100644
--- a/file1
+++ b/file1
@@ -1 +1 @@
-hello world
+Hello World
diff --git a/file2 b/file2
new file mode 100644
index 0000000..d77231c
--- /dev/null
+++ b/file2
@@ -0,0 +1 @@
+FILE2
```

Inspecting the History

```
$ git blame /file/to/blame
```

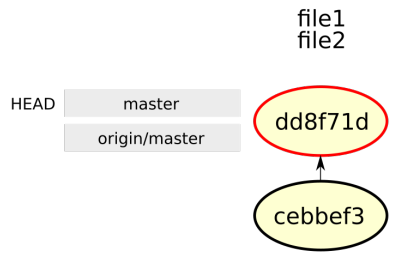
Inspecting the History

```
$ git blame /file/to/blame
```

```
$ git blame README
3773b454 (Michael Witten     2012-04-02 01:07:52 +0000   1)         Linux kernel release 3.x <http://kernel.org/
^1da177e (Linus Torvalds     2005-04-16 15:20:36 -0700   2)
0466dcbe (Jesper Juhl        2011-07-01 12:46:56 +0200   3) These are the release notes for Linux version 3.
Read them carefully,
^1da177e (Linus Torvalds     2005-04-16 15:20:36 -0700   4) as they tell you what this is all about, explain how
^1da177e (Linus Torvalds     2005-04-16 15:20:36 -0700   5) kernel, and what to do if something goes wrong.
^1da177e (Linus Torvalds     2005-04-16 15:20:36 -0700   6)
^1da177e (Linus Torvalds     2005-04-16 15:20:36 -0700   7) WHAT IS LINUX?
^1da177e (Linus Torvalds     2005-04-16 15:20:36 -0700   8)
4f4e2dc3 (Xose Vazquez Perez 2006-01-14 19:56:28 +0100   9)   Linux is a clone of the operating system Unix, wr
4f4e2dc3 (Xose Vazquez Perez 2006-01-14 19:56:28 +0100  10)   Linus Torvalds with assistance from a loosely-kni
4f4e2dc3 (Xose Vazquez Perez 2006-01-14 19:56:28 +0100  11)   the Net. It aims towards POSIX and Single UNIX Sp
^1da177e (Linus Torvalds     2005-04-16 15:20:36 -0700  12)
4f4e2dc3 (Xose Vazquez Perez 2006-01-14 19:56:28 +0100  13)   It has all the features you would expect in a mod
4f4e2dc3 (Xose Vazquez Perez 2006-01-14 19:56:28 +0100  14)   including true multitasking, virtual memory, shar
4f4e2dc3 (Xose Vazquez Perez 2006-01-14 19:56:28 +0100  15)   loading, shared copy-on-write executables, proper
4f4e2dc3 (Xose Vazquez Perez 2006-01-14 19:56:28 +0100  16)   and multistack networking including IPv4 and IPv6
^1da177e (Linus Torvalds     2005-04-16 15:20:36 -0700  17)
^1da177e (Linus Torvalds     2005-04-16 15:20:36 -0700  18)   It is distributed under the GNU General Public Li
^1da177e (Linus Torvalds     2005-04-16 15:20:36 -0700  19)   accompanying COPYING file for more details.
```

Undoing Changes

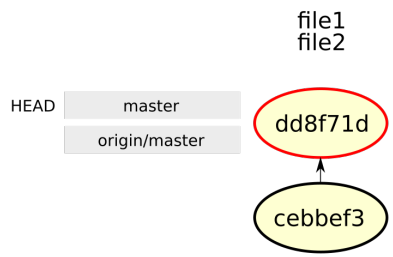
Changes tracked but **not yet added**



Undoing Changes

Changes tracked but **not yet added**

Warning changes will be lost forever...



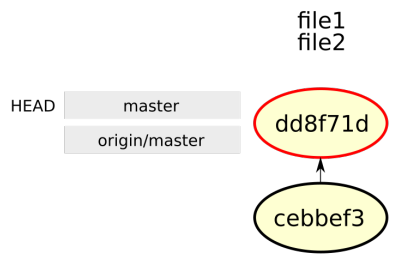
Undoing Changes

Changes tracked but **not yet added**

Warning changes will be lost forever...

Surgical undo:

```
$ git checkout -p -- /file/to/file
```



Undoing Changes

Changes tracked but **not yet added**

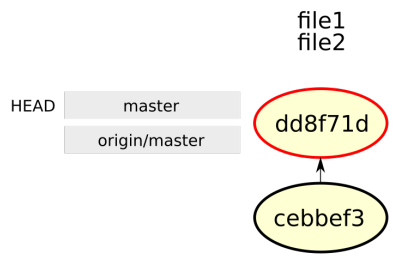
Warning changes will be lost forever...

Surgical undo:

```
$ git checkout -p -- /file/to/file
```

Undo one file:

```
$ git checkout -- /path/to/file
```



Undoing Changes

Changes tracked but **not yet added**

Warning changes will be lost forever...

Surgical undo:

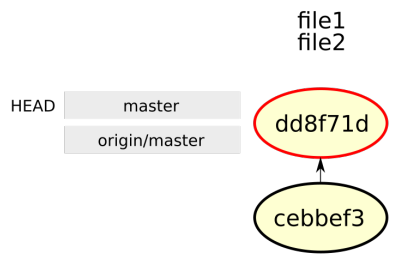
```
$ git checkout -p -- /file/to/file
```

Undo one file:

```
$ git checkout -- /path/to/file
```

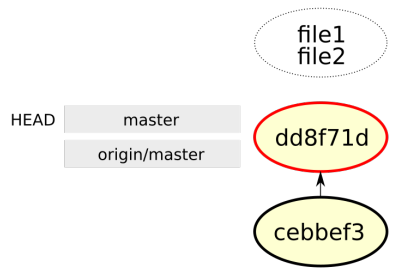
Nuclear blast:

```
$ git reset --hard
```



Undoing Changes

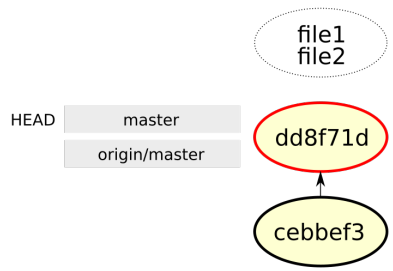
Changes added (`git add /path/to/file`),
but **not yet commited**



Undoing Changes

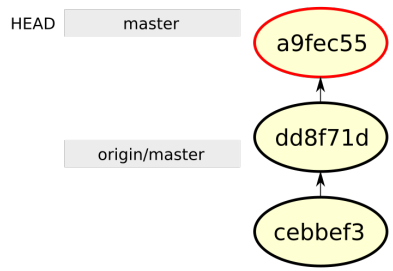
Changes added (`git add /path/to/file`), but **not yet commited**

```
$ git reset [-p] /path/to/added/file
```



Undoing Changes

Changes commited, but **not yet pushed**

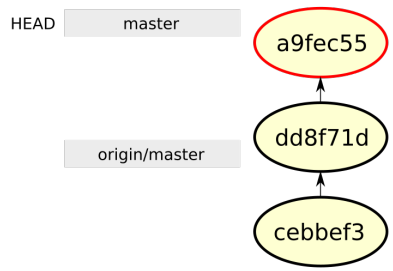


Undoing Changes

Changes commited, but **not yet pushed**

Fine grained:

```
$ git rebase -i SHA1
```



Undoing Changes

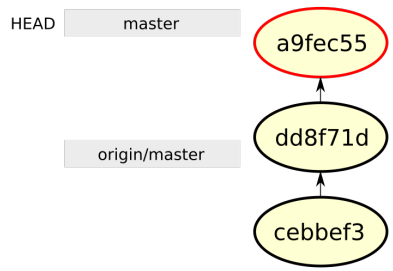
Changes commited, but **not yet pushed**

Fine grained:

```
$ git rebase -i SHA1
```

Blast the last commit:

```
$ git reset --hard HEAD^
$ git reset --hard SHA1
```



Undoing Changes

Changes commited, but **not yet pushed**

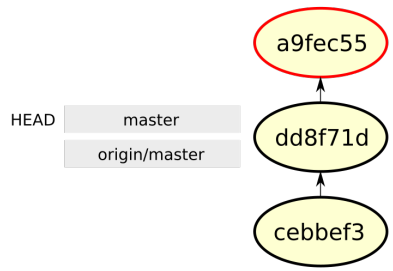
Fine grained:

```
$ git rebase -i SHA1
```

Blast the last commit:

```
$ git reset --hard HEAD^
$ git reset --hard SHA1
```

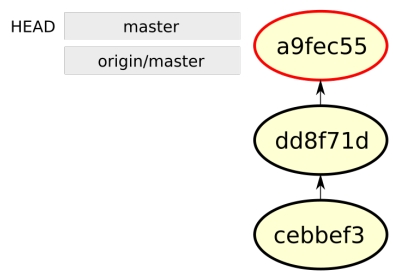
Commit is not deleted, but not easy to retrieve...



Undoing Changes

Changes **already pushed**

→ Never change a pushed commit

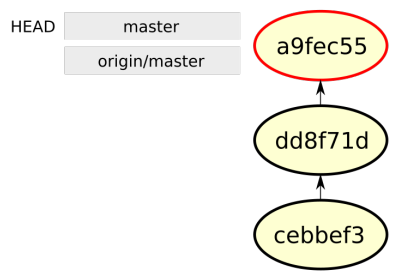


Undoing Changes

Changes **already pushed**

→ Never change a pushed commit

```
$ git revert SHA1
```

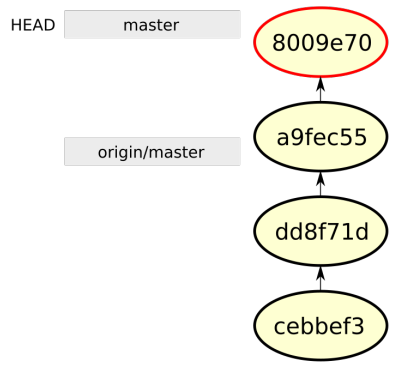


Undoing Changes

Changes **already pushed**

→ Never change a pushed commit

```
$ git revert SHA1
```

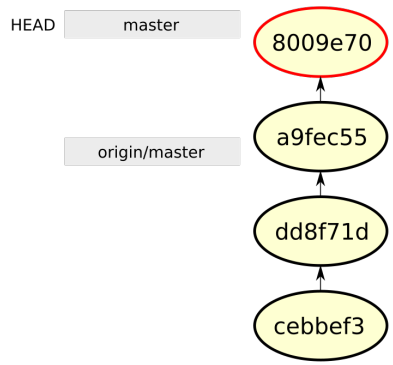


Undoing Changes

Changes **already pushed**

→ Never change a pushed commit

```
$ git revert SHA1
$ git log --oneline
```



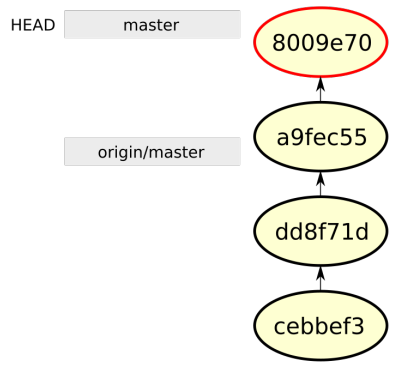
Undoing Changes

Changes **already pushed**

→ Never change a pushed commit

```
$ git revert SHA1
$ git log --oneline
```

```
$ git log --oneline
8009e70 Revert "Removing useless file"
7622a83 Removing useless file
dd8f71d Add readme
cebbef3 Initial commit
```

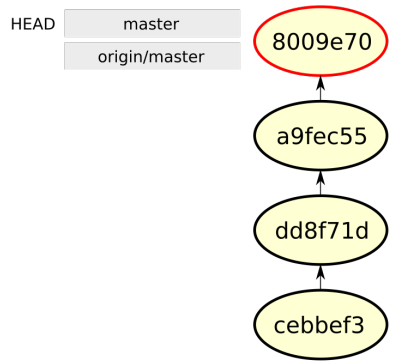


Undoing Changes

Changes **already pushed**

→ Never change a pushed commit

```
$ git revert SHA1
$ git log --oneline
$ git push
```



Undoing Changes

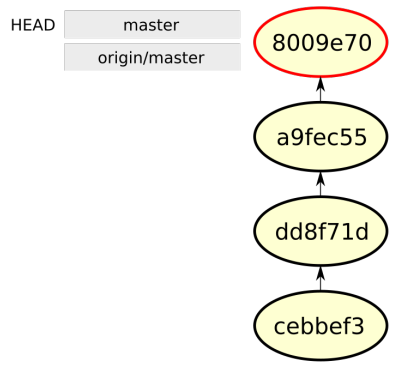
Changes **already pushed**

→ Never change a pushed commit

```
$ git revert SHA1
$ git log --oneline
$ git push
```

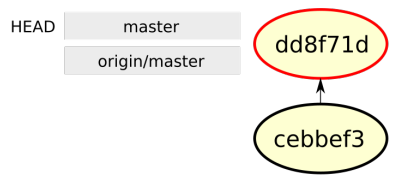
If you want to perform a partial revert

```
$ git revert SHA1 --no-commit
$ git reset
$ git add -p /file/to/revert
$ git commit
$ git reset --hard
```



Branches

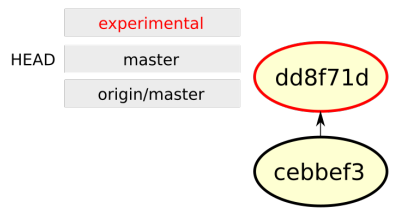
Creating a new branch



Branches

Creating a new branch

```
$ git branch experimental
```

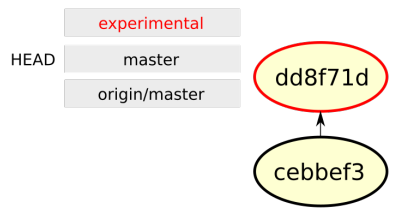


Branches

Creating a new branch

```
$ git branch experimental
$ git branch -v
```

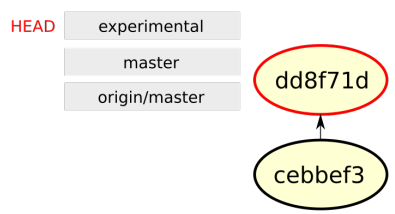
```
$ git branch -v
  experimental dd8f71d Removing useless file
* master       dd8f71d Removing useless file
```



Branches

Creating a new branch

```
$ git branch experimental
$ git branch -v
$ git checkout experimental
```



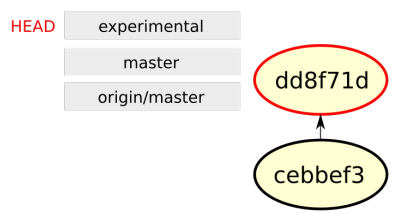
Branches

Creating a new branch

```
$ git branch experimental
$ git branch -v
$ git checkout experimental
```

or faster

```
$ git checkout -b experimental
```



Branches

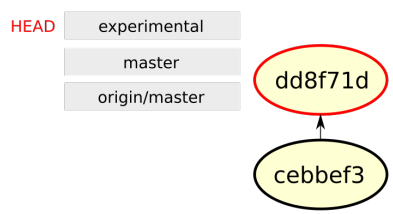
Creating a new branch

```
$ git branch experimental
$ git branch -v
$ git checkout experimental
```

or faster

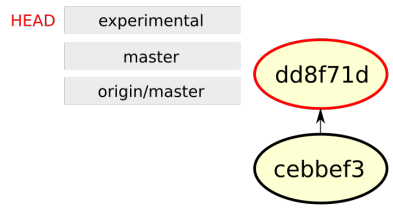
```
$ git checkout -b experimental
```

```
$ cat .git/HEAD
ref: refs/heads/experimental
$ cat .git/refs/heads/experimental
dd8f71d70f6b3710136fe9e393c8cecdf3839431
```



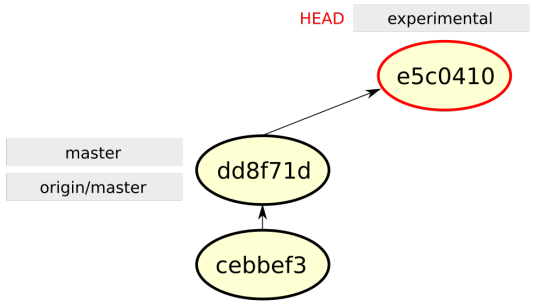
Branches

Make a new commit...



Branches

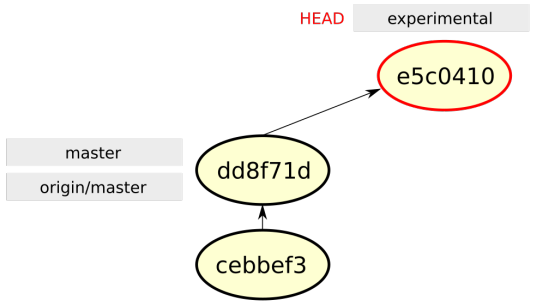
Make a new commit...



Branches

Make a new commit...
Make a new commit on branch 'master'...

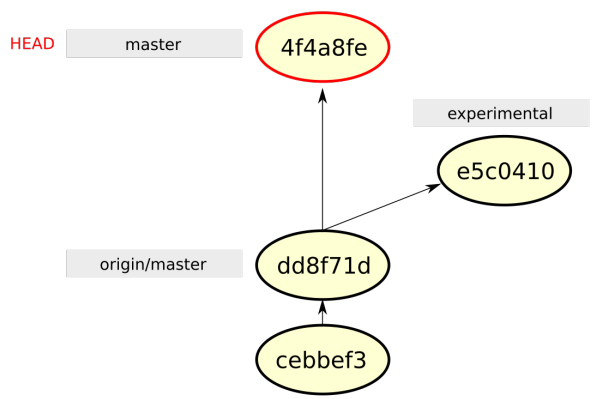
```
$ git checkout master
...
$ git commit
```



Branches

Make a new commit...
Make a new commit on branch 'master'...

```
$ git checkout master
...
$ git commit
```



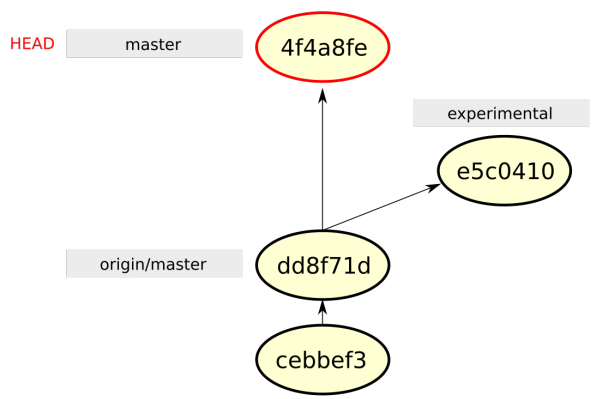
Branches

Make a new commit...
Make a new commit on branch 'master'...

```
$ git checkout master
...
$ git commit
```

Now you can merge if you are happy

```
$ git merge experimental
```



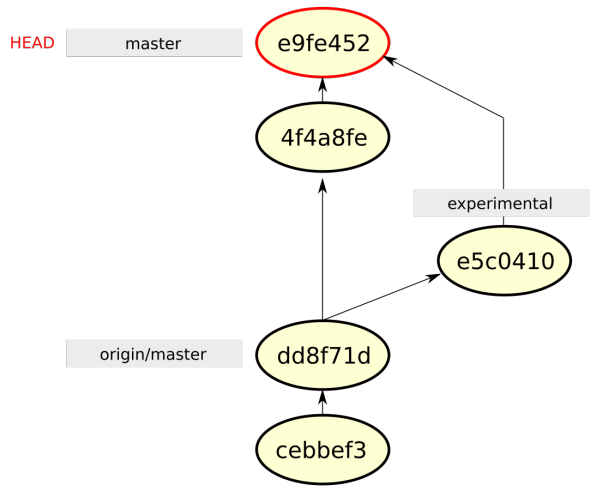
Branches

Make a new commit...
Make a new commit on branch 'master'...

```
$ git checkout master
...
$ git commit
```

Now you can merge if you are happy

```
$ git merge experimental
```



Branches

Make a new commit...
Make a new commit on branch 'master'...

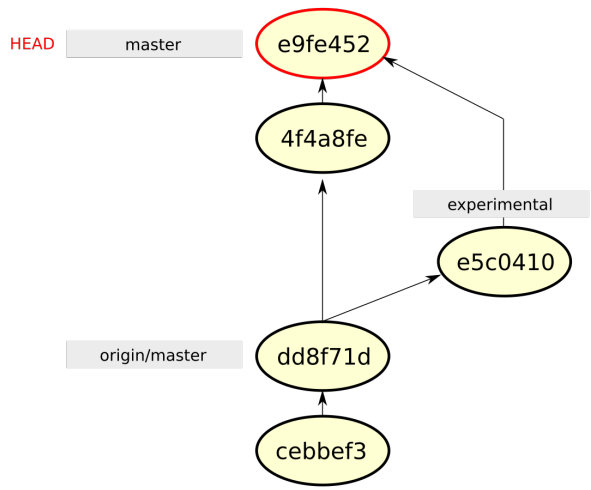
```
$ git checkout master
...
$ git commit
```

Now you can merge if you are happy

```
$ git merge experimental
```

And share your work

```
$ git push
```



Branches

Make a new commit...
Make a new commit on branch 'master'...

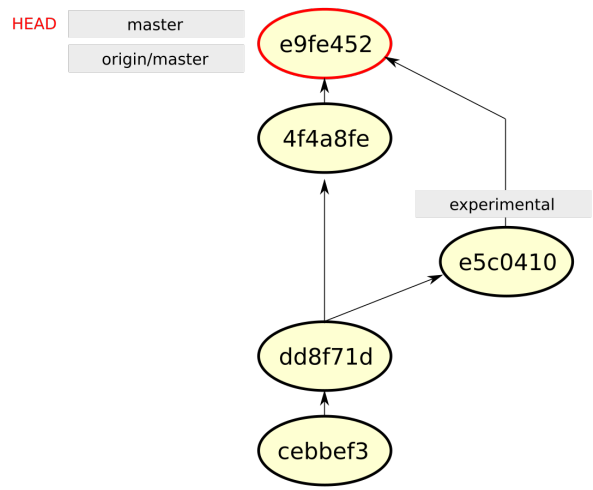
```
$ git checkout master
...
$ git commit
```

Now you can merge if you are happy

```
$ git merge experimental
```

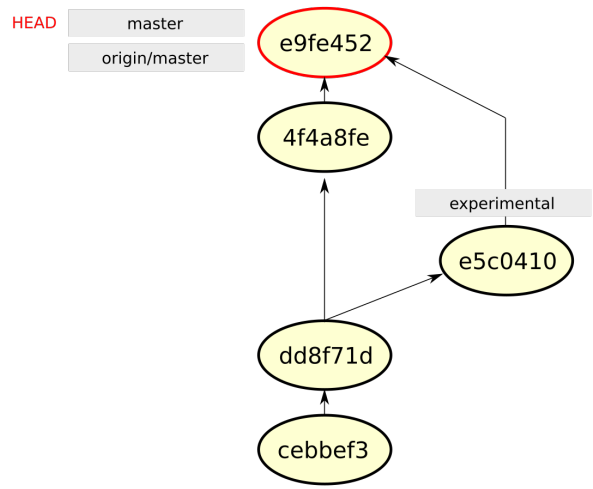
And share your work

```
$ git push
```



Branches

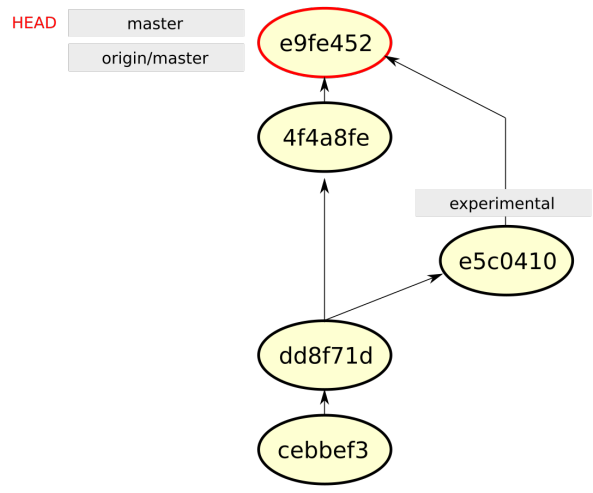
Deleting a local branch is as easy as



Branches

Deleting a local branch is as easy as

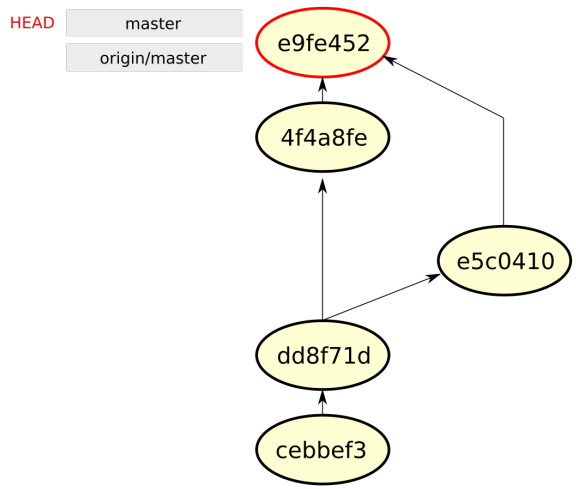
```
$ git branch -d experimental
```



Branches

Deleting a local branch is as easy as

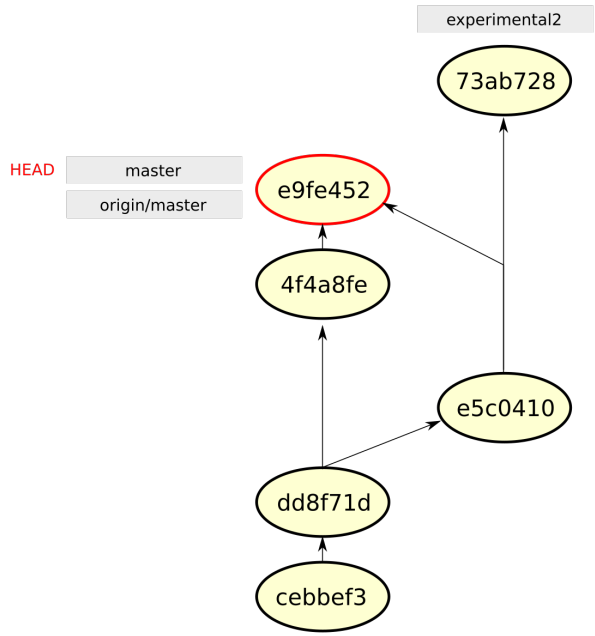
```
$ git branch -d experimental
```



Branches

Deleting a local branch is as easy as

```
$ git branch -d experimental
```



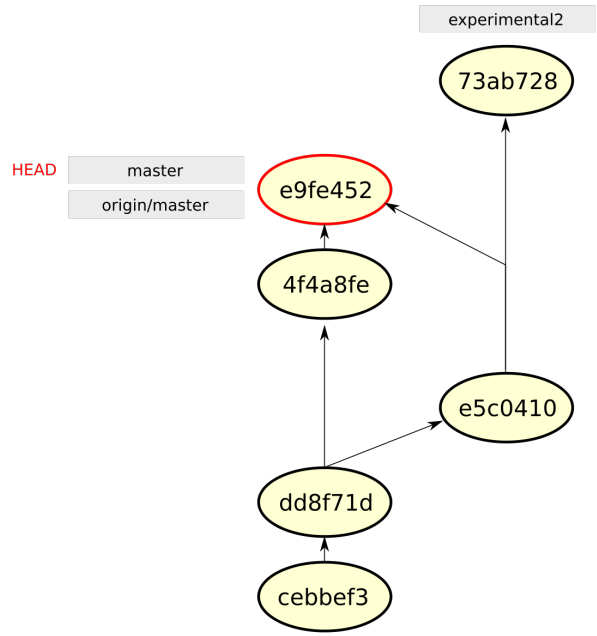
Branches

Deleting a local branch is as easy as

```
$ git branch –d experimental
```

Sometimes you need to insist a bit

```
$ git branch –D experimental2
```



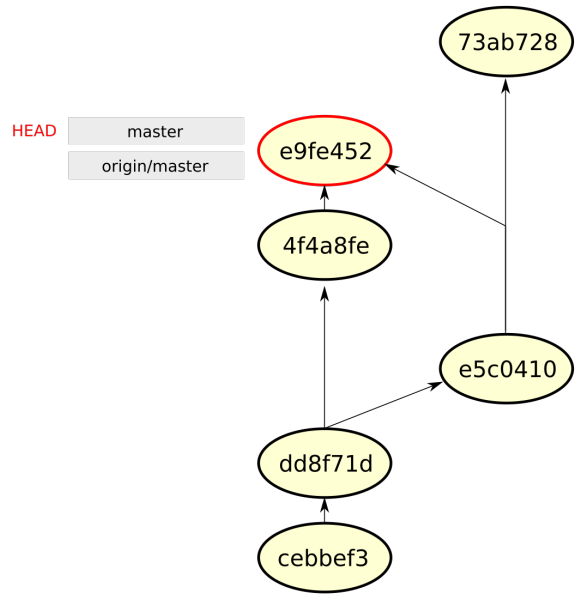
Branches

Deleting a local branch is as easy as

```
$ git branch -d experimental
```

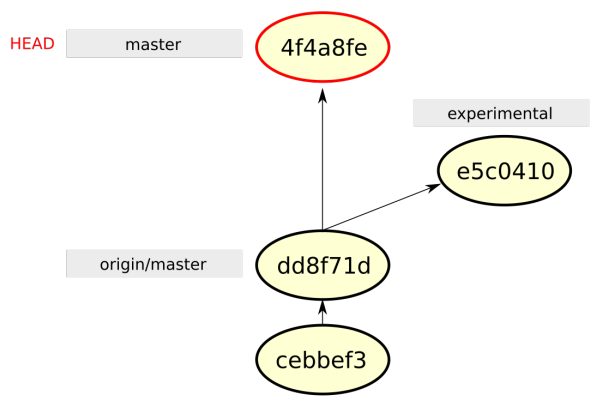
Sometimes you need to insist a bit

```
$ git branch -D experimental2
```



Branches

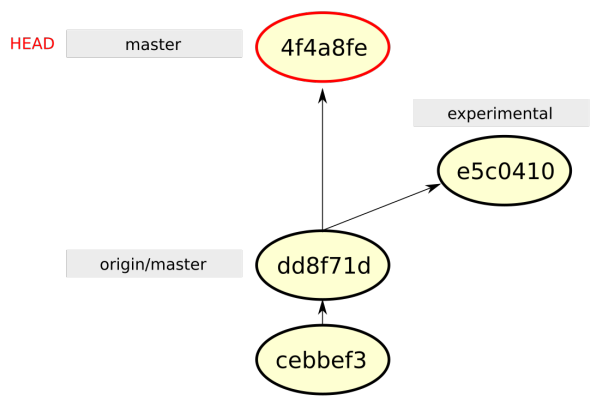
You can also push your topic branch to the server



Branches

You can also push your topic branch to the server

```
$ git push origin branch-name
```

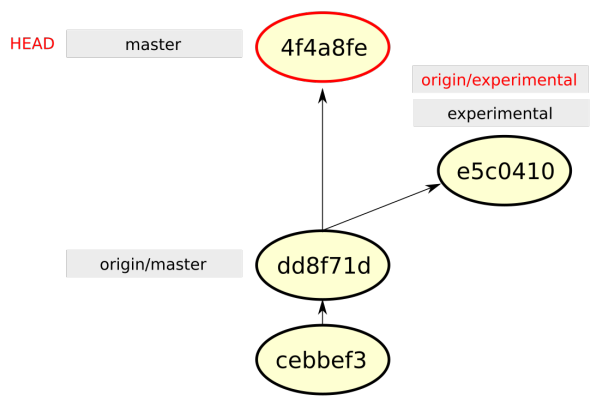


Branches

You can also push your topic branch to the server

```
$ git push origin branch-name
```

```
$ git push origin experimental
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 345 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@redsmine.heig-vd.ch:project
 * [new branch]      experimental -> experimental
```



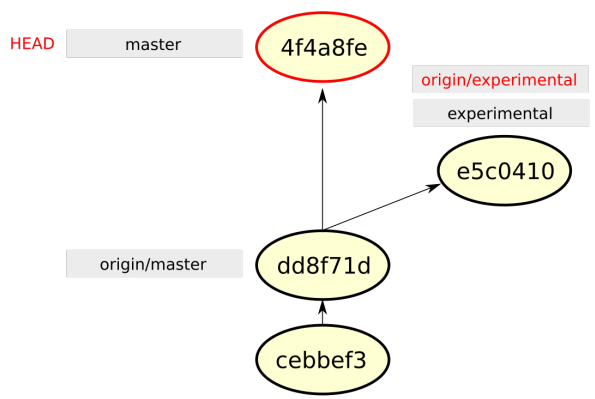
Branches

You can also push your topic branch to the server

```
$ git push origin branch-name
```

To delete it from the server

```
$ git push origin :remote-branch-name
```



Branches

You can also push your topic branch to the server

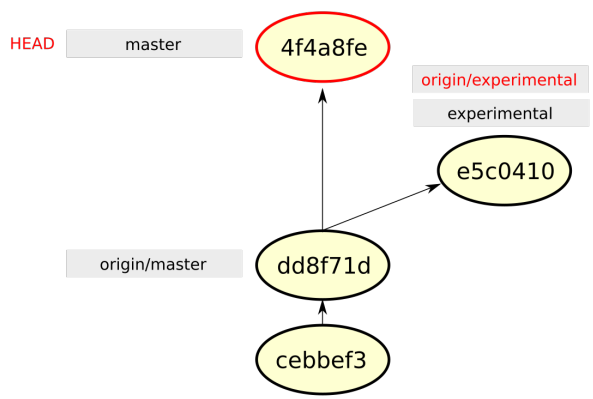
```
$ git push origin branch-name
```

To delete it from the server

```
$ git push origin :remote-branch-name
```

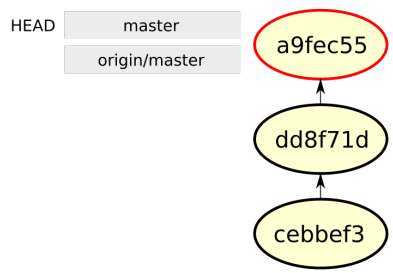
General form of `git push`

```
$ git push server <src>:<dst>
```



Branches

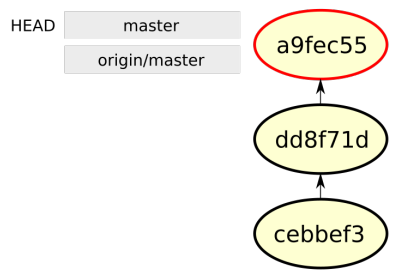
And you can get branches from others!



Branches

And you can get branches from others!

```
$ git fetch origin
```

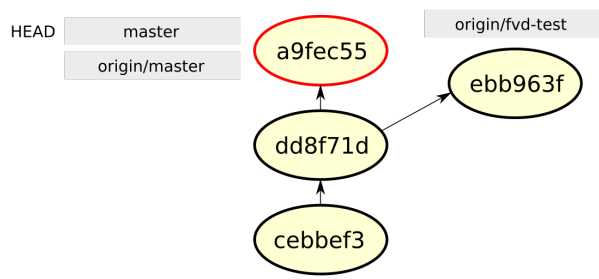


Branches

And you can get branches from others!

```
$ git fetch origin
```

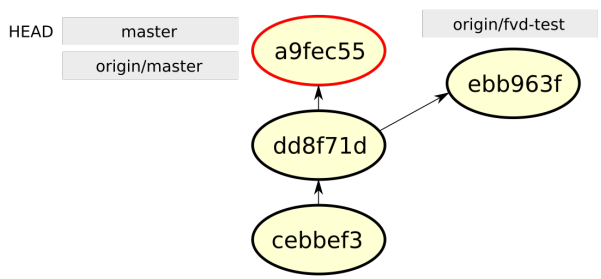
```
$ git fetch origin
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From git@redsmine.heig-vd.ch:project
 * [new branch]      fvd-test   -> origin/fvd-test
```



Branches

And you can get branches from others!

```
$ git fetch origin
$ git branch -a
```

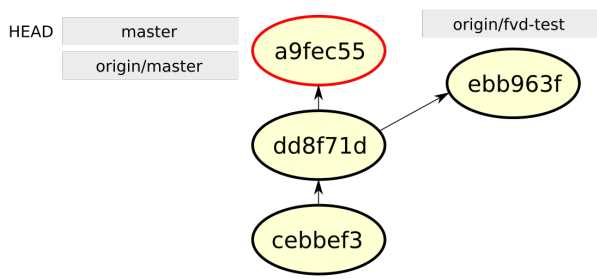


Branches

And you can get branches from others!

```
$ git fetch origin
$ git branch -a
```

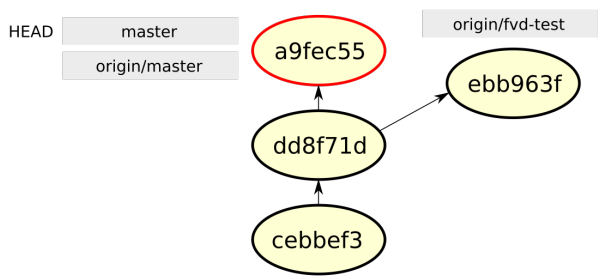
```
$ git branch -a
* master
  remotes/origin/fvd-test
```



Branches

And you can get branches from others!

```
$ git fetch origin
$ git branch -a
$ git checkout -b test origin/fvd-test
```

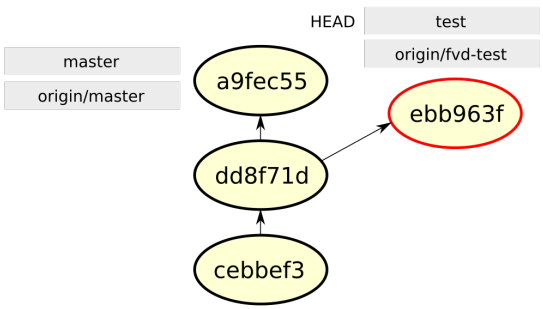


Branches

And you can get branches from others!

```
$ git fetch origin
$ git branch -a
$ git checkout -b test origin/fvd-test
```

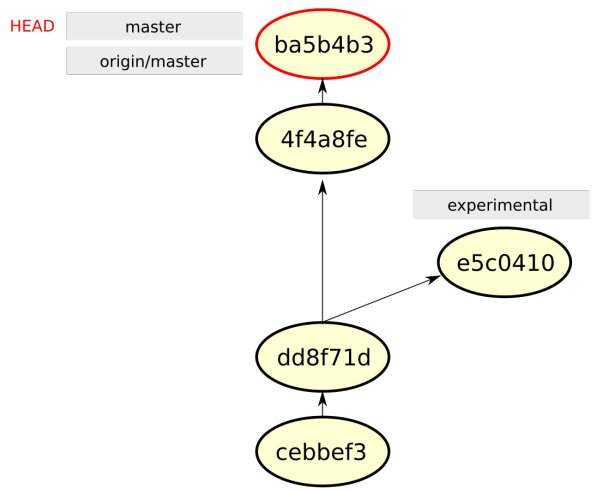
```
$ git checkout -b test fvd-test
Branch test set up to track remote branch fvd-test from origin.
Switched to a new branch 'test'
```



Branches

Keeping Up with Upstream

Time passes...
It is time to keep up with upstream!

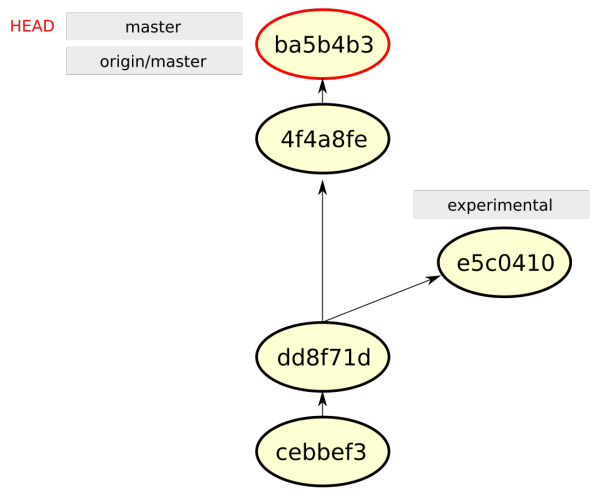


Branches

Keeping Up with Upstream

Time passes...
It is time to keep up with upstream!

Let's rebase branch 'experimental' on 'master'



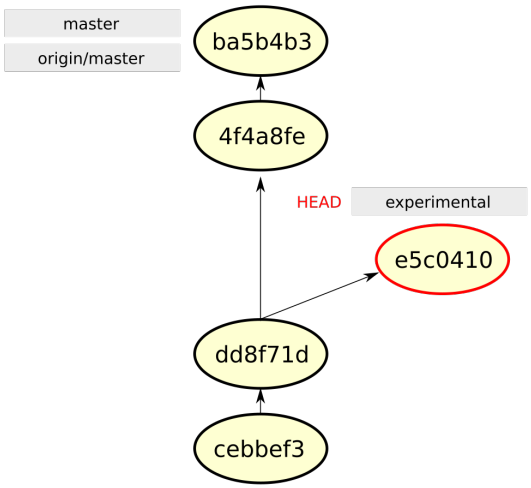
Branches

Keeping Up with Upstream

Time passes...
It is time to keep up with upstream!

Let's rebase branch 'experimental' on 'master'

```
$ git checkout experimental
```



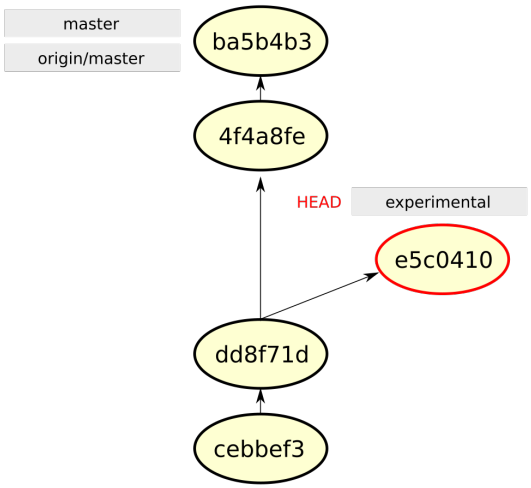
Branches

Keeping Up with Upstream

Time passes...
It is time to keep up with upstream!

Let's rebase branch 'experimental' on 'master'

```
$ git checkout experimental

$ git rebase master
```



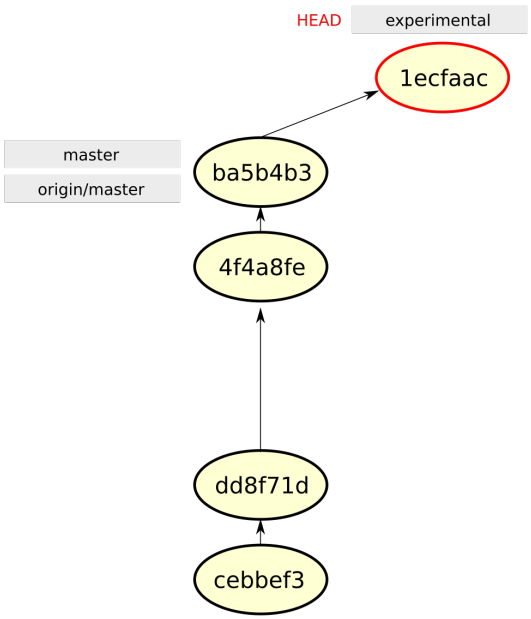
Branches

Keeping Up with Upstream

Time passes...
It is time to keep up with upstream!

Let's rebase branch 'experimental' on 'master'

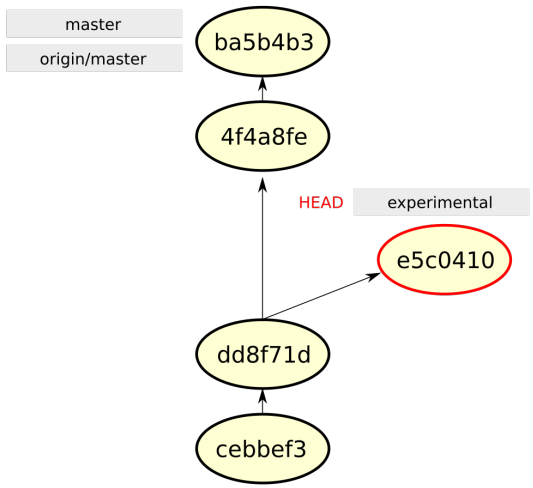
```
$ git checkout experimental

$ git rebase master
```



Branches

Cherry Picking

Let's copy-n-paste!

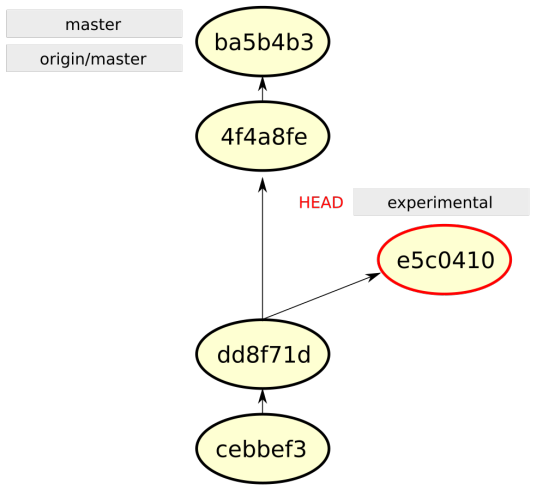


Branches

Cherry Picking

Let's copy-n-paste!

```
$ git cherry-pick 4f4a8fe
```

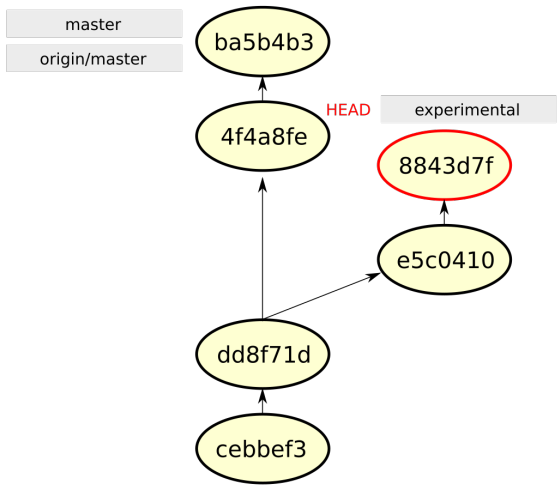


Branches

Cherry Picking

Let's copy-n-paste!

```
$ git cherry-pick 4f4a8fe
```



Branches

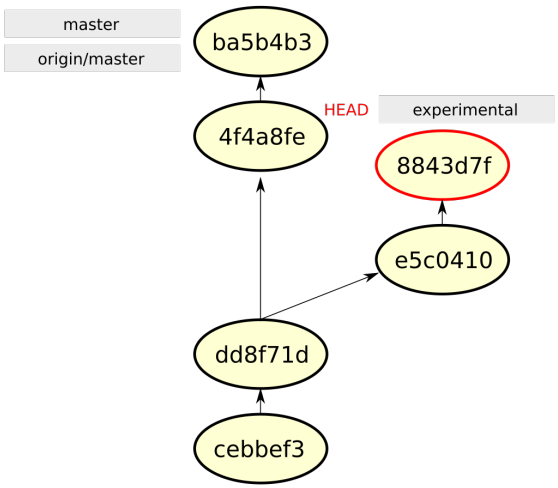
Cherry Picking

Let's copy-n-paste!

```
$ git cherry-pick 4f4a8fe
```

There is no link between both commits!
→ We lose traceability :-(

(but you can use `git cherry-pick -x`)



Branches

Cherry Picking

Let's copy-n-paste!

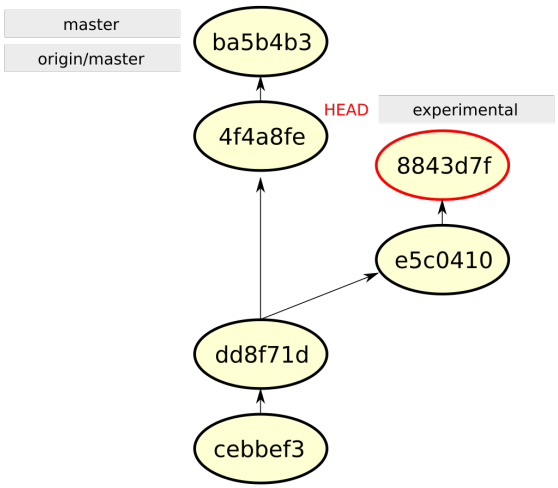
```
$ git cherry-pick 4f4a8fe
```

There is no link between both commits!
→ We lose traceability :-(

(but you can use `git cherry-pick -x`)

Cherry pick multiple commits

```
$ git cherry-pick SHA1 SHA1 SHA1
$ git cherry-pick <start>..<end>
```



Tags

Tags are attached to one specific commit ($\neq$ branches)

Tags

Tags are attached to one specific commit ($\neq$ branches)

Tag the current commit

```
$ git tag -a v1.4 -m "Version 1.4 is out!"
```

Tags

Tags are attached to one specific commit ($\neq$ branches)

Tag the current commit

```
$ git tag -a v1.4 -m "Version 1.4 is out!"
```

Tag a previous commit

```
$ git tag -a v1.2 -m "Version 1.2 is out!" SHA1
```

Tags

Tags are attached to one specific commit ($\neq$ branches)

Tag the current commit

```
$ git tag -a v1.4 -m "Version 1.4 is out!"
```

Tag a previous commit

```
$ git tag -a v1.2 -m "Version 1.2 is out!" SHA1
```

Push your tags

```
$ git push origin --tags
```

Tags

Tags are attached to one specific commit ($\neq$ branches)

Tag the current commit

```
$ git tag -a v1.4 -m "Version 1.4 is out!"
```

Tag a previous commit

```
$ git tag -a v1.2 -m "Version 1.2 is out!" SHA1
```

Push your tags

```
$ git push origin --tags
```

Why using tags?

Tags

Tags are attached to one specific commit ($\neq$ branches)

Tag the current commit

```
$ git tag -a v1.4 -m "Version 1.4 is out!"
```

Tag a previous commit

```
$ git tag -a v1.2 -m "Version 1.2 is out!" SHA1
```

Push your tags

```
$ git push origin --tags
```

Why using tags?

- Place milestones

Tags

Tags are attached to one specific commit ($\neq$ branches)

Tag the current commit

```
$ git tag -a v1.4 -m "Version 1.4 is out!"
```

Tag a previous commit

```
$ git tag -a v1.2 -m "Version 1.2 is out!" SHA1
```

Push your tags

```
$ git push origin --tags
```

Why using tags?

- Place milestones
- Easily checkout a commit

What is a Good Commit?

Moto

Commit early, commit often

What is a Good Commit?

Moto

Commit early, commit often

Each commit should be

- Small (one feature at a time)

What is a Good Commit?

Moto

Commit early, commit often

Each commit should be

- Small (one feature at a time)
- Be self contained

What is a Good Commit?

Moto

Commit early, commit often

Each commit should be

- Small (one feature at a time)
- Be self contained
- Compile & work

What is a Good Commit?

Why ?

What is a Good Commit?

Why ?

- `git log`

What is a Good Commit?

Why ?

- `git log`
- `git blame`

What is a Good Commit?

Why ?

- `git log`
- `git blame`
- `git revert`

What is a Good Commit?

Why ?

- `git log`
- `git blame`
- `git revert`
- `git bisect`

What is a Good Commit?

How ?

What is a Good Commit?

How ?

- Commit early, commit often!

What is a Good Commit?

How ?

- Commit early, commit often!
- `git status` and `git diff`

What is a Good Commit?

How ?

- Commit early, commit often!
- `git status` and `git diff`
- `git add [-p] /file/to/add`
  (no `git add -u` and never `git add -a`)

What is a Good Commit?

How ?

- Commit early, commit often!
- `git status` and `git diff`
- `git add [-p] /file/to/add`
  (no `git add -u` and never `git add -a`)
- `git diff --cached`

Outline

Make your Life Easier

aliases

You can configure 'aliases' (shortcuts) in your `~/.gitconfig`

Make your Life Easier

aliases

You can configure 'aliases' (shortcuts) in your `~/.gitconfig`

```
$ cat ~/.gitconfig
[user]
        name = Florian Vaussard
        email = florian.vaussard@heig-vd.ch
[alias]
        cout = checkout
        b = branch -v
        lol = log --oneline --decorate=short
```

Referencing a Commit

Many commands take a revision as parameter

- A single revision (e.g. `git show dae86e`)
- A range (e.g. `git log v4.4..v4.5`)

Referencing a Commit

Many commands take a revision as parameter

- A single revision (e.g. `git show dae86e`)
- A range (e.g. `git log v4.4..v4.5`)

Examples of a single revision

- sha1: dae86e1950b1277e545cee180551750029cfe735; dae86e
- refname: branch name (local or remote); tag name; HEAD
- refname@{date}: master@{yesterday}; HEAD@{5 minutes ago}
- rev^: HEAD^
- rev~n: master~3

Referencing a Commit

Examples of a range

- rev: `git log v4.3`
- rev1..rev2: `git log v4.3..v4.5`

Referencing a Commit

Examples of a range

- rev: `git log v4.3`
- rev1..rev2: `git log v4.3..v4.5`

For more information

```
$ man gitrevisions
```

Partial Add

You can add only part of a file with `git add -p`

Partial Add

You can add only part of a file with `git add -p`

Example

```
$ git diff
diff --git a/file1 b/file1
index 3b18e51..c3b969f 100644
--- a/file1
+++ b/file1
@@ -1 +1,9 @@
+Title
+=====
+
 hello world
+
+Resources
+---------
+
+[1] http://foobar.net
```

Partial Add

```
$ git add -p file1
```

Partial Add

```
$ git add -p file1
diff --git a/file1 b/file1
index 3b18e51..c3b969f 100644
--- a/file1
+++ b/file1
@@ -1 +1,9 @@
+Title
+=====
+
 hello world
+
+Resources
+---------
+
+[1] http://foobar.net
Stage this hunk [y,n,q,a,d,/,s,e,?]?
```

Partial Add

```
Stage this hunk [y,n,q,a,d,/,s,e,?]? h
y - stage this hunk
n - do not stage this hunk
q - quit; do not stage this hunk or any of the remaining ones
a - stage this hunk and all later hunks in the file
d - do not stage this hunk or any of the later hunks in the file
g - select a hunk to go to
/ - search for a hunk matching the given regex
j - leave this hunk undecided, see next undecided hunk
J - leave this hunk undecided, see next hunk
k - leave this hunk undecided, see previous undecided hunk
K - leave this hunk undecided, see previous hunk
s - split the current hunk into smaller hunks
e - manually edit the current hunk
? - print help
```

Partial Add

Split!

```
Stage this hunk [y,n,q,a,d,/,s,e,?]? s
Split into 2 hunks.
@@ -1 +1,4 @@
+Title
+======
+
  hello world
```

Partial Add

Split!

```
Stage this hunk [y,n,q,a,d,/,s,e,?]? s
Split into 2 hunks.
@@ -1 +1,4 @@
+Title
+======
+
  hello world

Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? y
```

Partial Add

```
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? y
@@ -1 +4,6 @@
 hello world
+
+Resources
+---------
+
+[1] http://foobar.net
```

Partial Add

```
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? y
@@ -1 +4,6 @@
 hello world
+
+Resources
+----------
+
+[1] http://foobar.net

Stage this hunk [y,n,q,a,d,/,K,g,e,?]? n
```

Partial Add

```
$ git diff --cached
```

Partial Add

```
$ git diff --cached
diff --git a/file1 b/file1
index 3b18e51..d79c347 100644
--- a/file1
+++ b/file1
@@ -1 +1,4 @@
+Title
+======
+
 hello world
```

Partial Add

```
$ git diff --cached
diff --git a/file1 b/file1
index 3b18e51..d79c347 100644
--- a/file1
+++ b/file1
@@ -1 +1,4 @@
+Title
+======
+
 hello world
```

Let's commit...

```
$ git commit -m "Add a nice title"
```

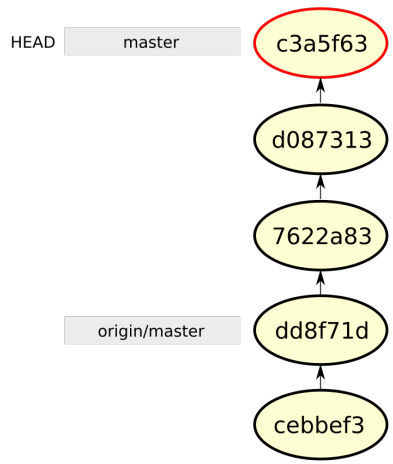
Interactive Rebase

Git lets you rewrite the history.

Interactive Rebase

Git lets you rewrite the history.

Warning Never ever change the history of pushed commits!

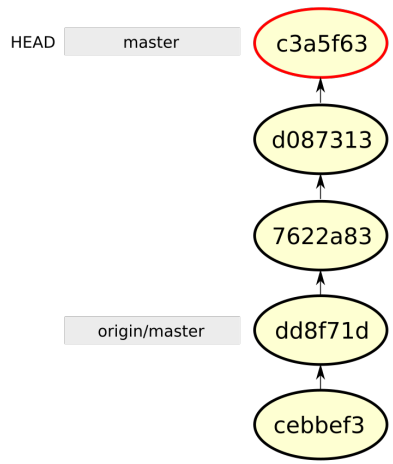


Interactive Rebase

Git lets you rewrite the history.

Warning Never ever change the history of pushed commits!

```
$ git rebase -i <base-point>
```

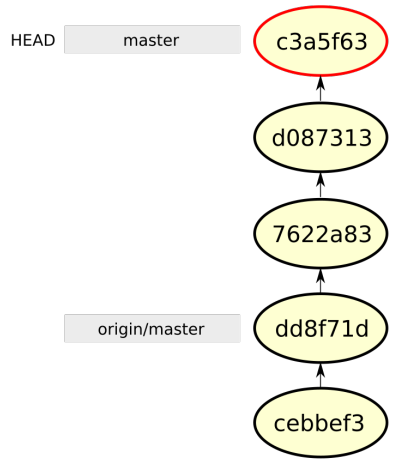


Interactive Rebase

Git lets you rewrite the history.

Warning Never ever change the history of pushed commits!

```
$ git rebase -i <base-point>

e.g. git rebase -i origin/master
```

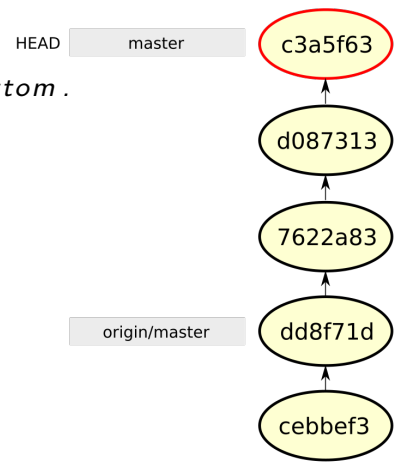


Interactive Rebase

Git lets you rewrite the history.

Warning Never ever change the history of pushed commits!

```
pick 7622a83 Removing useless file
pick d087313 Add file3
pick c3a5f63 Add file4

# Rebase dd8f71d..c3a5f63 onto dd8f71d (3 command(s))
#
# 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
# x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

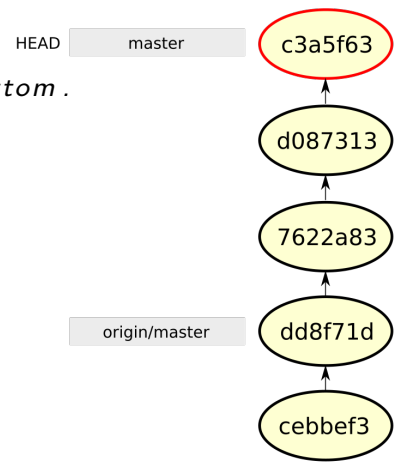


Interactive Rebase

Git lets you rewrite the history.

Warning Never ever change the history of pushed commits!

```
pick 7622a83 Removing useless file
pick c3a5f63 Add file4
pick d087313 Add file3

# Rebase dd8f71d..c3a5f63 onto dd8f71d (3 command(s))
#
# 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
# x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

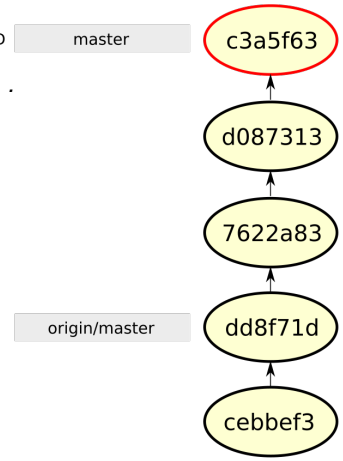


Interactive Rebase

Git lets you rewrite the history.

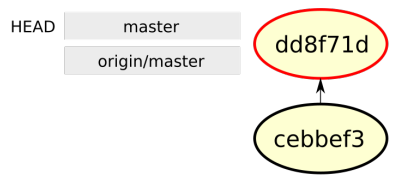
Warning Never ever change the history of pushed commits!

```
pick 7622a83 Removing useless file
f c3a5f63 Add file4
r d087313 Add file3

# Rebase dd8f71d..c3a5f63 onto dd8f71d (3 command(s))
#
# 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
# x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```



Working with Remotes

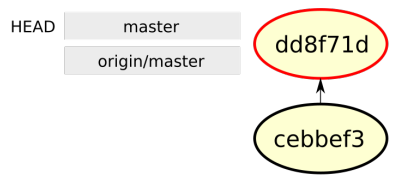
Git is a distributed revision control system.



Working with Remotes

Git is a distributed revision control system.

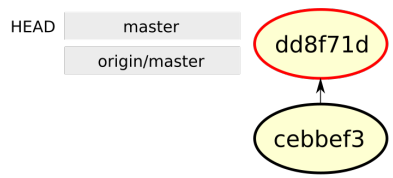
```
$ git remote -v
```



Working with Remotes

Git is a distributed revision control system.

```
$ git remote -v
origin  git@redsmine.heig-vd.ch:my-project (fetch)
origin  git@redsmine.heig-vd.ch:my-project (push)
```

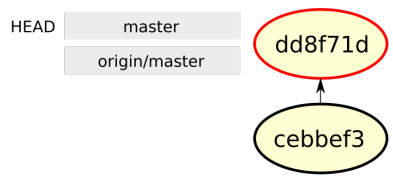


Working with Remotes

Git is a distributed revision control system.

```
$ git remote -v
```

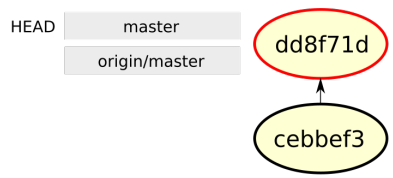
```
$ git remote add bob https://github.com/bob/my-project.git
```



Working with Remotes

Git is a distributed revision control system.

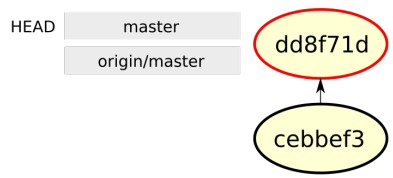
```
$ git remote -v

$ git remote add bob https://github.com/bob/my-project.git
$ git remote -v
origin   git@redsmine.heig-vd.ch:my-project (fetch)
origin   git@redsmine.heig-vd.ch:my-project (push)
bob   https://github.com/bob/my-project.git (fetch)
bob   https://github.com/bob/my-project.git (push)
```



Working with Remotes

Git is a distributed revision control system.

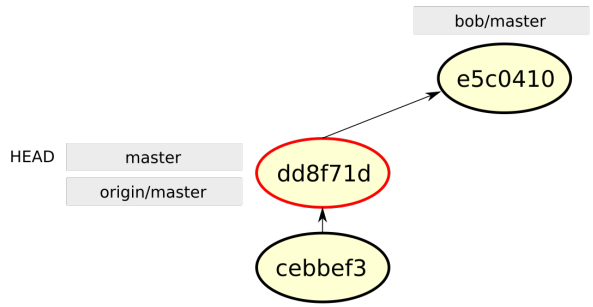
```
$ git remote -v

$ git remote add bob https://github.com/bob/my-project.git

$ git fetch bob
```



Working with Remotes

Git is a distributed revision control system.

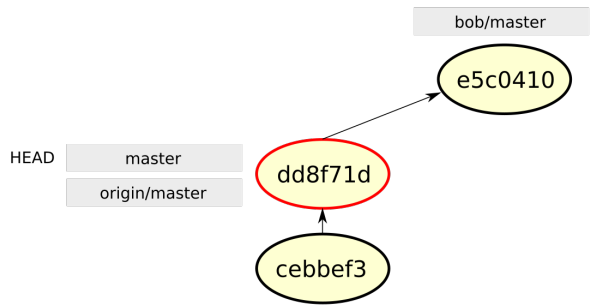
```
$ git remote -v

$ git remote add bob https://github.com/bob/my-project.git

$ git fetch bob
```



Working with Remotes

Git is a distributed revision control system.

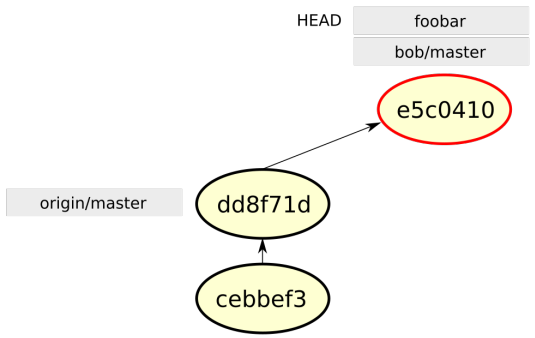
```
$ git remote -v

$ git remote add bob https://github.com/bob/my-project.git

$ git fetch bob

$ git log master..bob/master
```



Working with Remotes

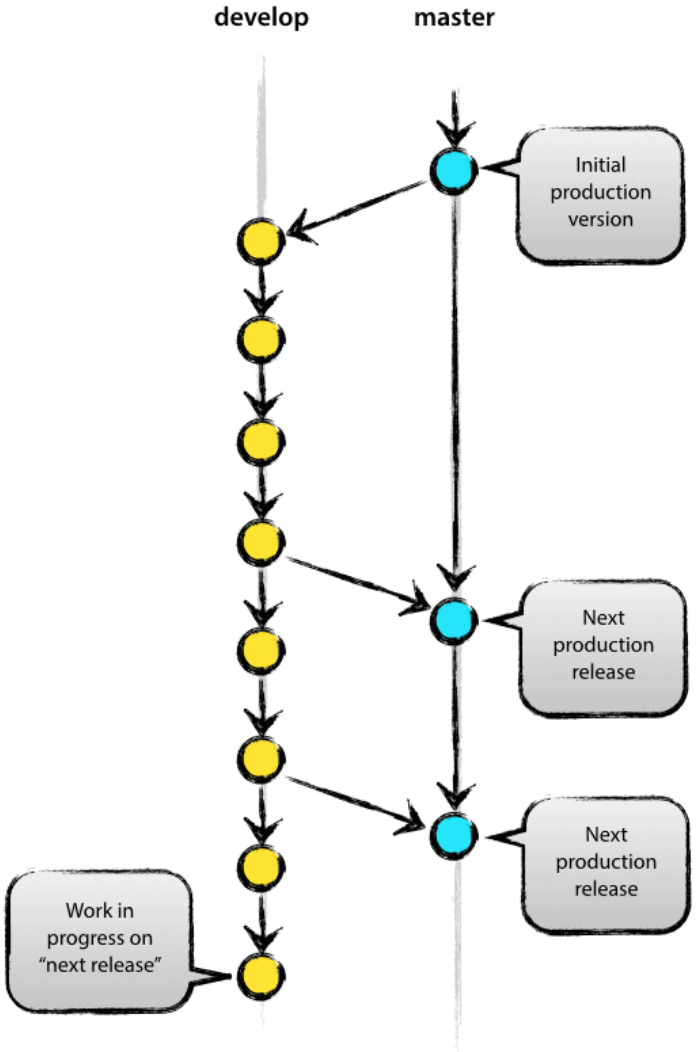
Git is a distributed revision control system.

```
$ git remote -v

$ git remote add bob https://github.com/bob/my-project.git

$ git fetch bob

$ git log master..bob/master

$ git checkout -b foobar bob/master
```



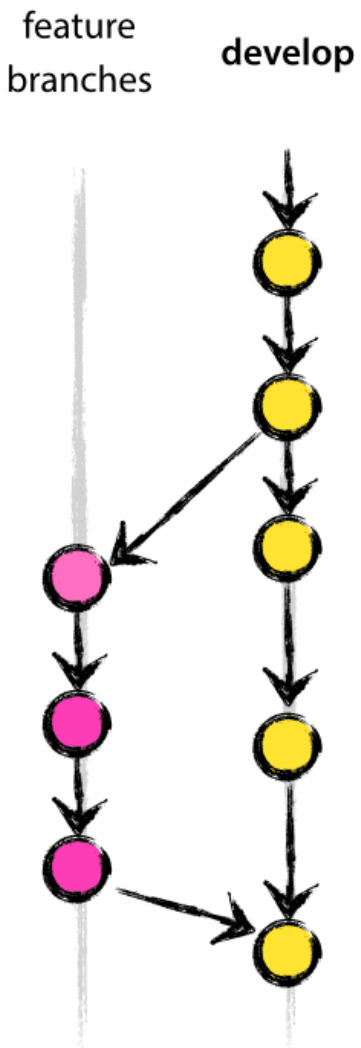
Typical Workflows

Simple Project



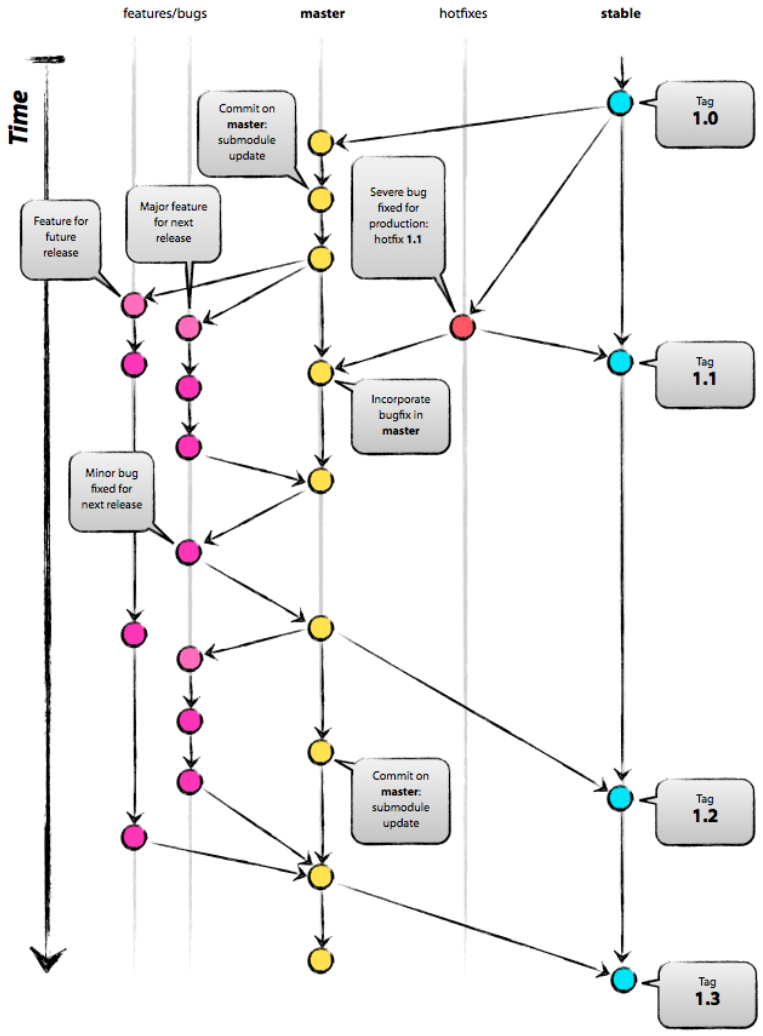
Typical Workflows

Simple Project



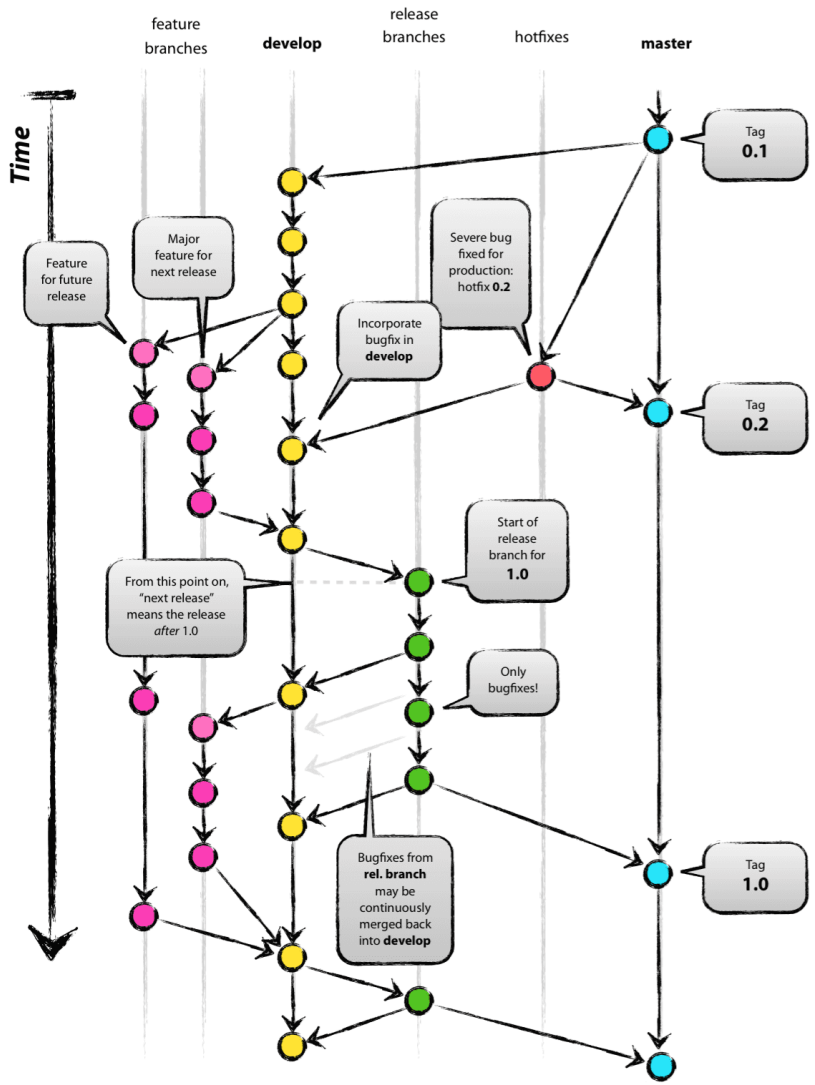
Typical Workflows

Complex Project



Typical Workflows

Complex Project



Other Useful Stuff

Git submodules

`git bisect`

Create patches

```
$ git format-patch
$ git send-email
```

Create .tar/.zip

```
$ git archive REF
```

Which version is it?

```
$ git describe --contains SHA1
```

When you completely messed up

```
$ git reflog
```

And many more...

Outline

Conclusion

To conclude

Conclusion

To conclude

- It can take years to learn all Git commands

Conclusion

To conclude

- It can take years to learn all Git commands
- Only a handful commands to master you have

Conclusion

To conclude

- It can take years to learn all Git commands
- Only a handful commands to master you have
  - `git branch` / `git checkout`

Conclusion

To conclude

- It can take years to learn all Git commands
- Only a handful commands to master you have
  - `git branch` / `git checkout`
  - `git status` / `git diff`

Conclusion

To conclude

- It can take years to learn all Git commands
- Only a handful commands to master you have
  - `git branch` / `git checkout`
  - `git status` / `git diff`
  - `git add [-p]` / `git commit`

Conclusion

To conclude

- It can take years to learn all Git commands
- Only a handful commands to master you have
  - `git branch` / `git checkout`
  - `git status` / `git diff`
  - `git add [-p]` / `git commit`
  - `git log` / `git show`

Conclusion

To conclude

- It can take years to learn all Git commands
- Only a handful commands to master you have
  - `git branch` / `git checkout`
  - `git status` / `git diff`
  - `git add [-p]` / `git commit`
  - `git log` / `git show`
  - `git fetch` / `git pull` / `git push`

Conclusion

To conclude

- It can take years to learn all Git commands
- Only a handful commands to master you have
  - `git branch` / `git checkout`
  - `git status` / `git diff`
  - `git add [-p]` / `git commit`
  - `git log` / `git show`
  - `git fetch` / `git pull` / `git push`
- For all the rest, StackOverflow is your best friend

Conclusion

To conclude

- It can take years to learn all Git commands
- Only a handful commands to master you have
  - `git branch` / `git checkout`
  - `git status` / `git diff`
  - `git add [-p]` / `git commit`
  - `git log` / `git show`
  - `git fetch` / `git pull` / `git push`
- For all the rest, StackOverflow is your best friend
- Use aliases to make your life easier