GIT

Trener: Maciej Krajewski

Gdańsk, 2 lipca 2016 roku

Agenda

VCS - co to jest po co się stosuje

GIT

- dlaczego powstał
- jak działa
- podstawowe operacje
- śledzenie historii
- gitignore

GIT a WEBStorm

GIT Good Practice

- co i kiedy commitować?
- nazewnictwo komentarzy

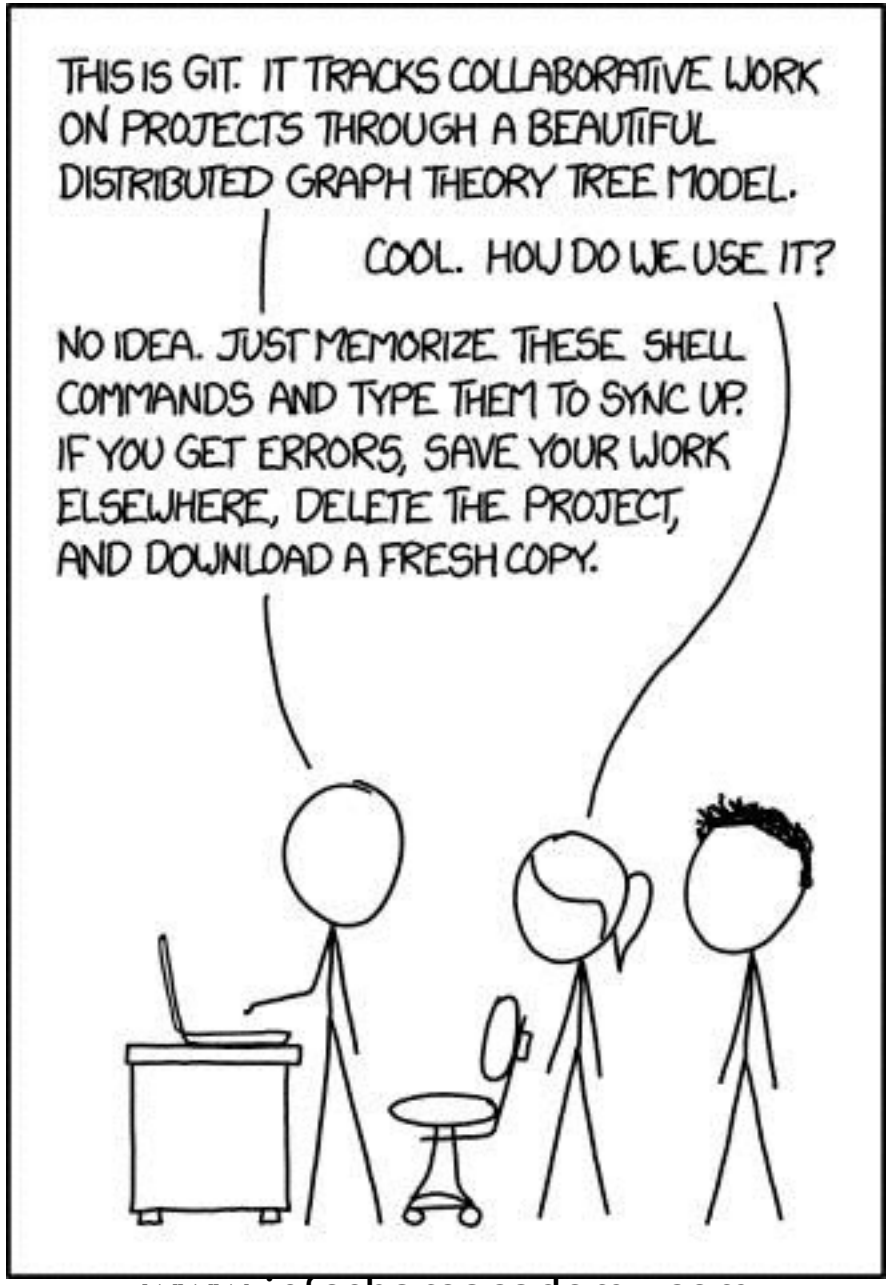
GIT cd

- rozwiązywanie konfliktów
- workflow w GIT
- git pull-request

VCS - co to jest po co się stosuje

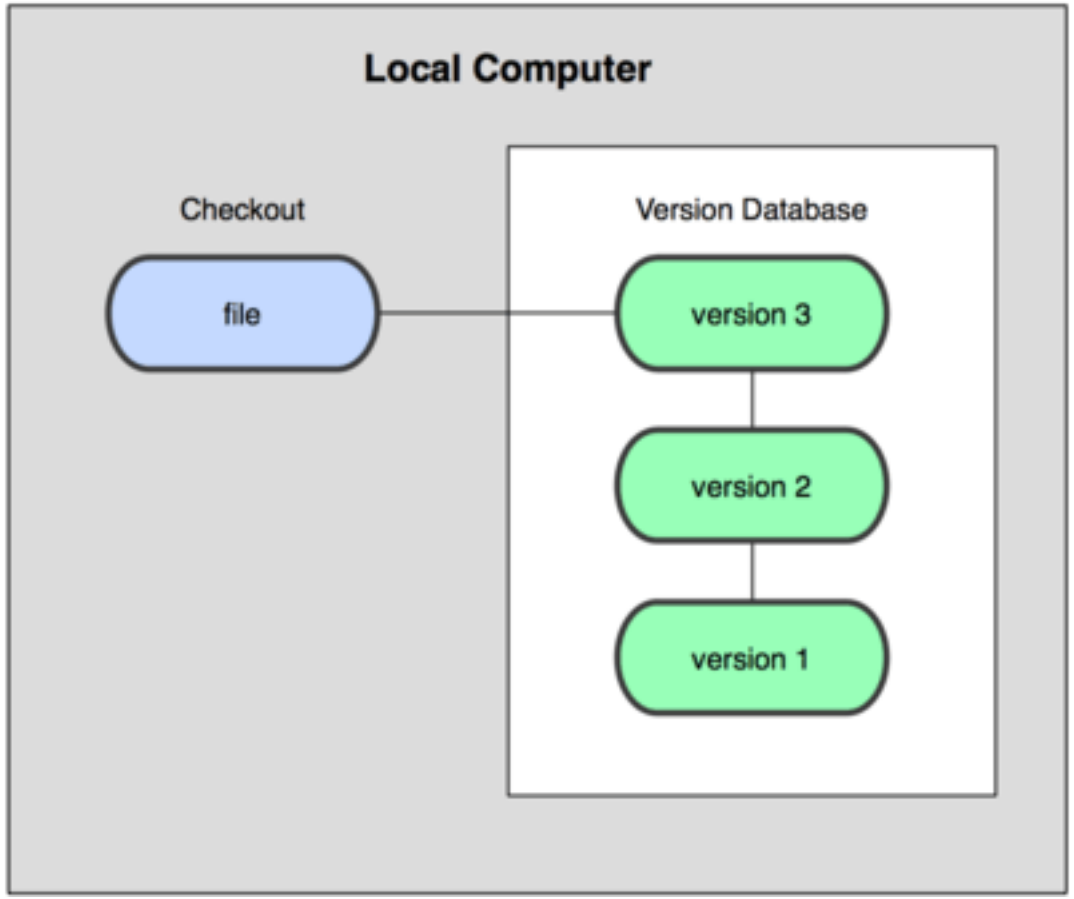
VCS’y odpowiadają podstawowym potrzebom IT:

- śledzenie zmian w kodzie (wersjonowanie)
- ułatwia współdzielenie tego samego kodu pomiędzy wieloma developerami*


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

VCS - co to jest po co się stosuje

System lokalny



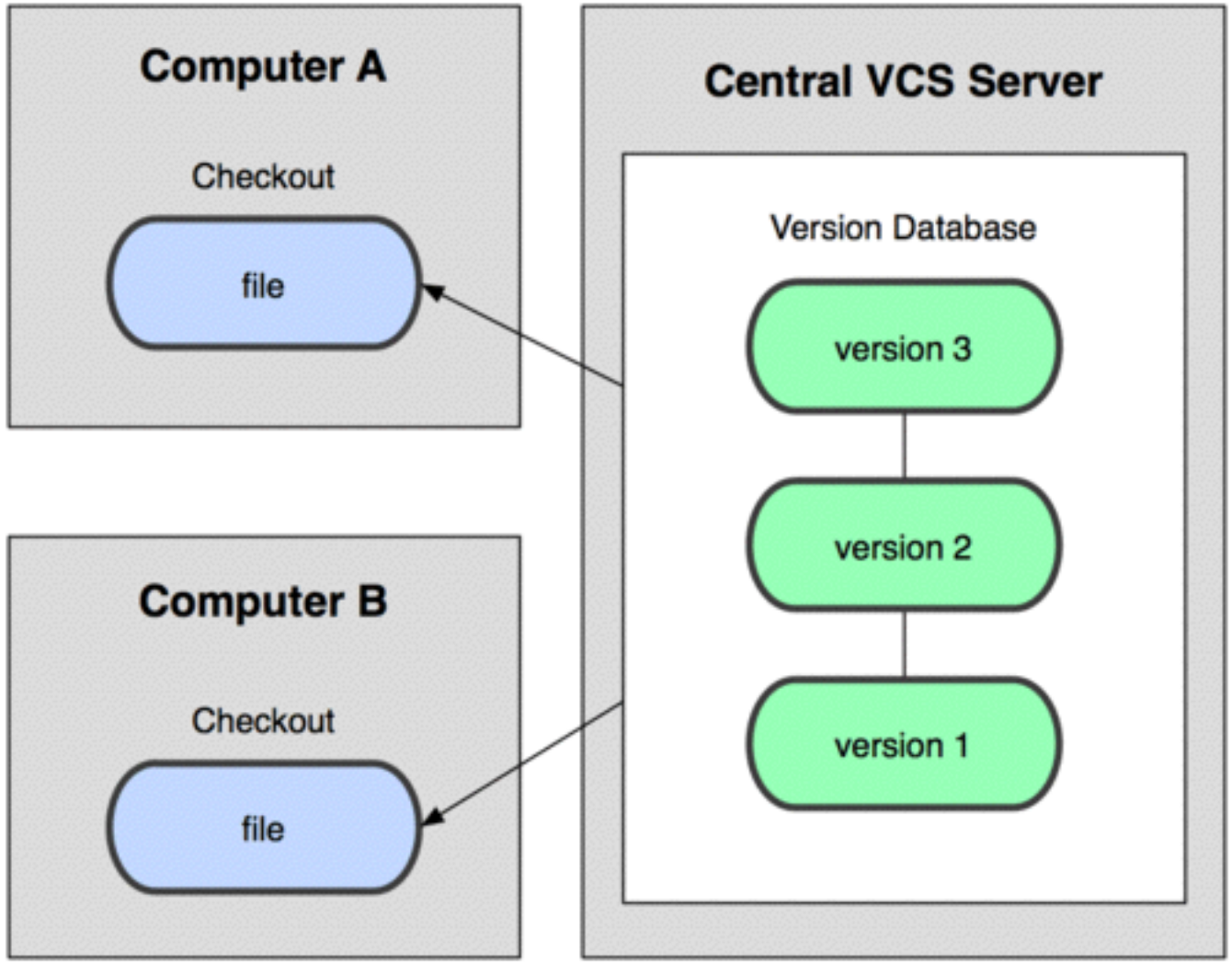
VCS - co to jest po co się stosuje

System scentralizowany

+ Umożliwia współdzielenie kodu pomiędzy innych developerów. (+)

- Każda zmiana w kodzie musi być wypchnięta i ściągnięta

- Konflikty



VCS - co to jest po co się stosuje

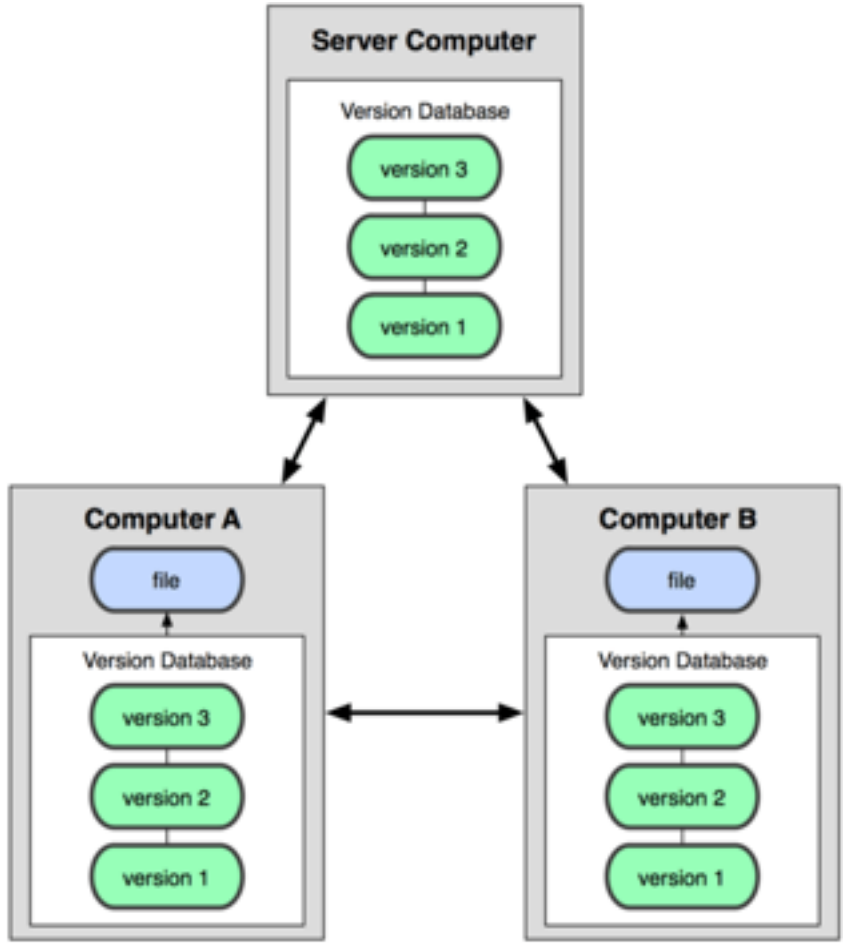
System rozproszony

+ Może być używany jako lokalny

+ Nie musimy mieć dostępu do serwera centralnego aby pracować

+ Operacje na repozytorium są szybsze

- Konflikty
- Problem spójności danych



GIT - dlaczego powstał
aka GIT - the stupid content tracker

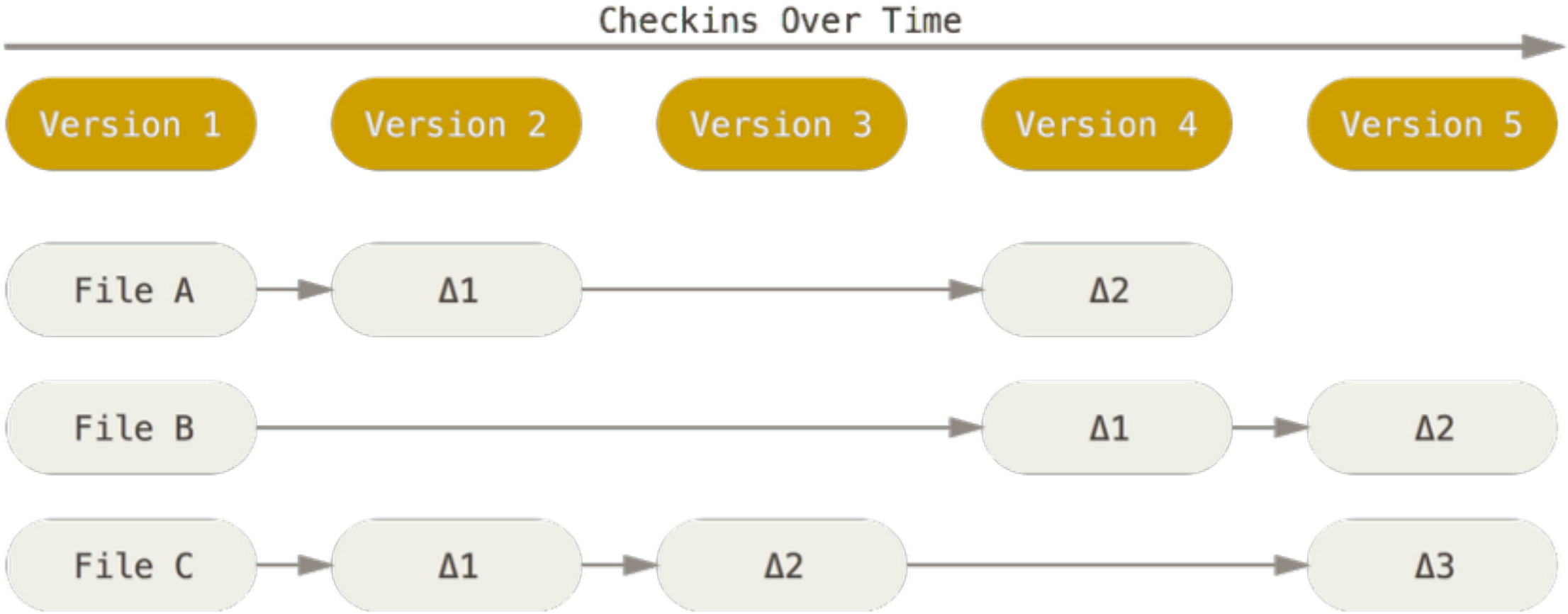
GIT powstał jako zapotrzebowanie projektu Open Source (jądra linuxa), po tym jak dotychczas używany BitKeeper przestał być darmowy dla projektów z otwartym kodem źródłowym.

Linus Torvald szukał systemu kontroli wersji, który spełniałby następujące wymagania:

1. Wziąć przykład z CVS, czego *nie* robić
2. System powinien być rozproszony
3. System powinien być chroniony przed błędami w repozytorium (przypadkowymi, jak awaria twardego dysku, jak i złośliwymi, wprowadzonymi przez kogoś)
4. System powinien być szybki

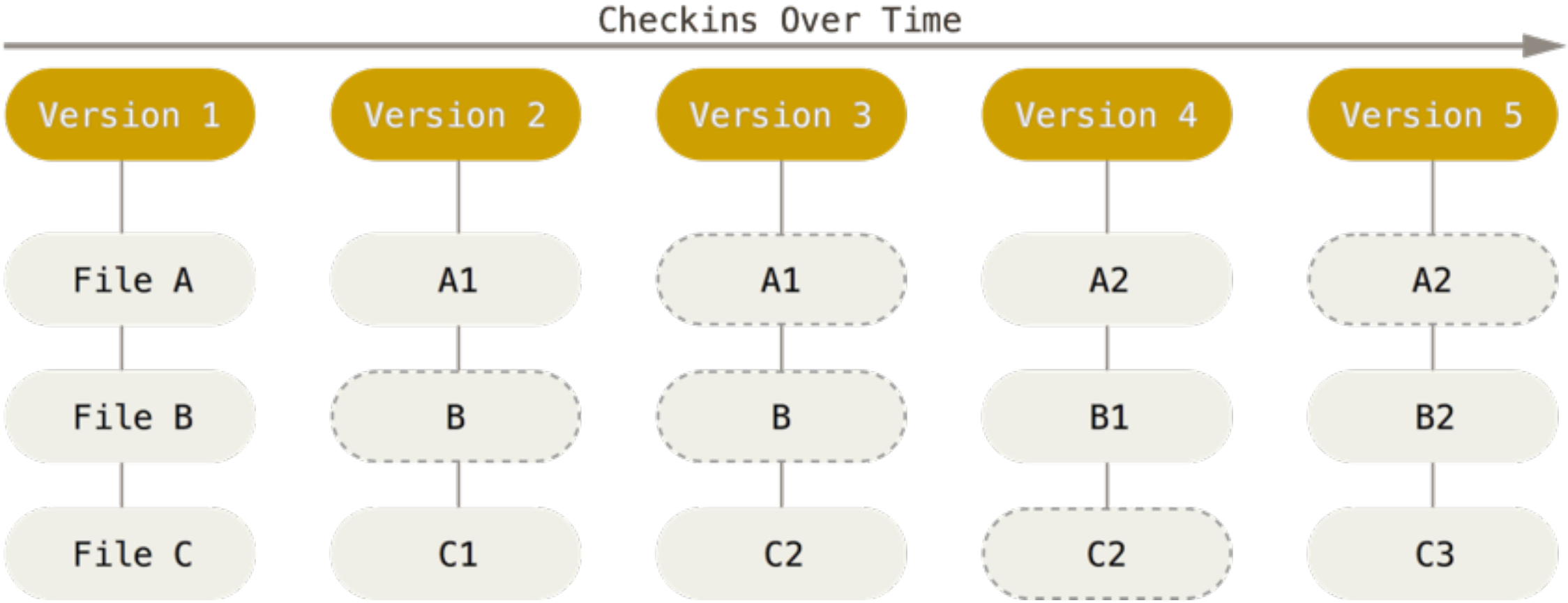
GIT - jak działa

Przechowywanie plików - sposób różnicowy - czyli jak GIT nie działa



GIT - jak działa

Przechowywanie plików - migawki - tak to jest plik



GIT - jak działa

Wszystkie operacje w GIT wykonywane są lokalnie.
Dzięki temu jest po prostu szybki!

GIT - jak działa

źródło: https://git-scm.com/about/small-and-fast

GIT - jak działa

Operation		Git	SVN	
Commit Files (A)	Add, commit and push 113 modified files (2164+, 2259-)	0.64	2.60	4x
Commit Images (B)	Add, commit and push 1000 1k images	1.53	24.70	16x
Diff Current	Diff 187 changed files (1664+, 4859-) against last commit	0.25	1.09	4x
Diff Recent	Diff against 4 commits back (269 changed/3609+,6898-)	0.25	3.99	16x
Diff Tags	Diff two tags against each other (v1.9.1.0/v1.9.3.0)	1.17	83.57	71x
Log (50)	Log of the last 50 commits (19k of output)	0.01	0.38	31x
Log (All)	Log of all commits (26,056 commits - 9.4M of output)	0.52	169.20	325x
Log (File)	Log of the history of a single file (array.c - 483 revs)	0.60	82.84	138x
Update	Pull of Commit A scenario (113 files changed, 2164+, 2259-)	0.90	2.82	3x
Blame	Line annotation of a single file (array.c)	1.91	3.04	1x

źródło: https://git-scm.com/about/small-and-fast

GIT - jak działa

Operation		Git*	Git	SVN
Clone	Clone and shallow clone(*) in Git vs checkout in SVN	21.0	107.5	14.0
Size (M)	Size of total client side data and files after clone/checkout (in M)		181.0	132.0

źródło: https://git-scm.com/about/small-and-fast

GIT - jak działa

GIT ma wbudowany mechanizm kontroli spójności danych. Dla każdego commitu wyliczana jest suma kontrolna SHA-1 zawierająca 40 znaków w systemie szesnastkowym.

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

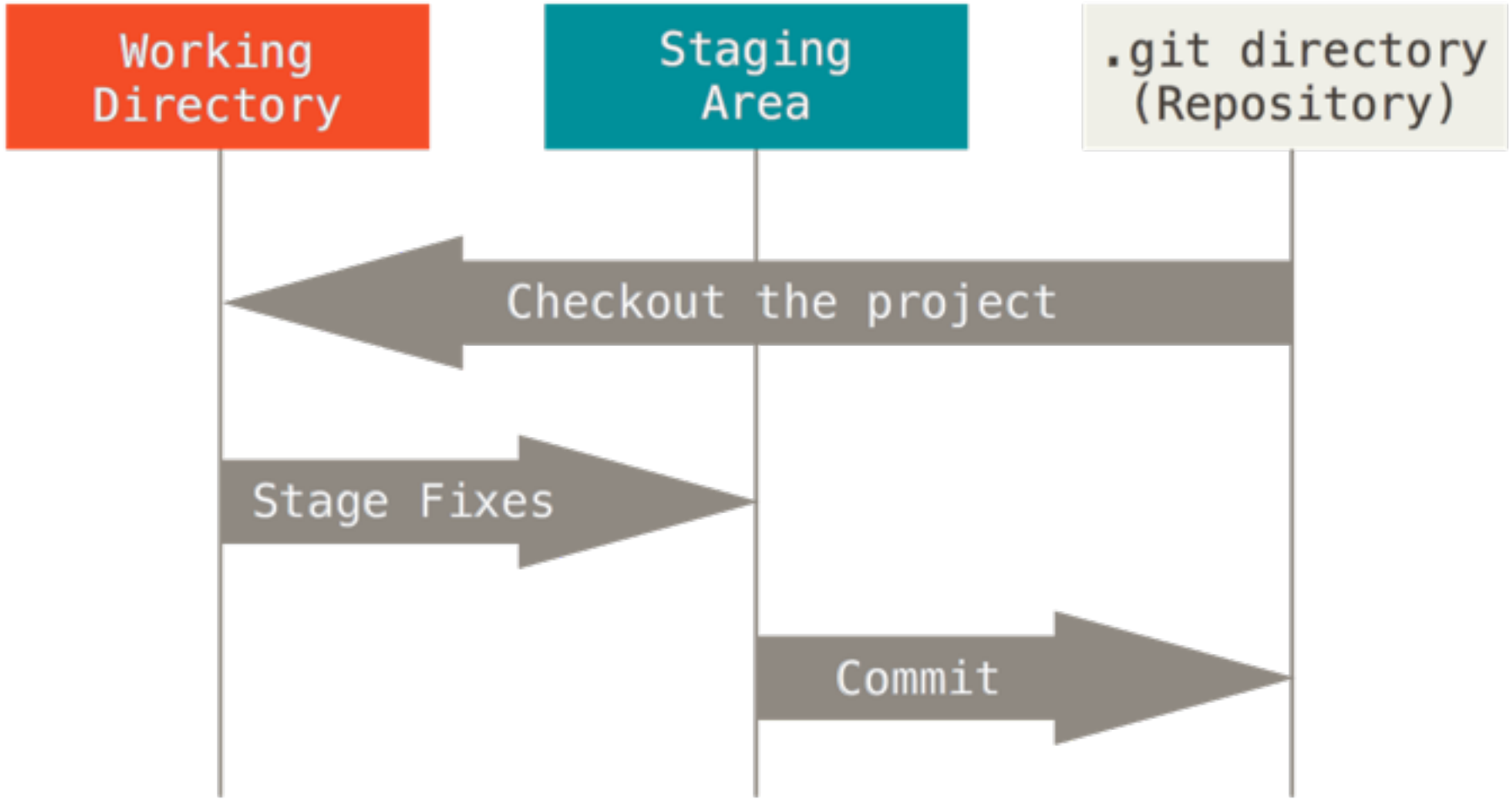
GIT - jak działa

GIT głównie dodaje tylko dane do lokalnego repozytorium.

Większość operacji w GIT dodaje nowe dane do repozytorium, dzięki czemu zmiany w kodzie zarządzanym przez GIT możemy uważać za bezpieczne, ponieważ w każdym momencie możemy odtworzyć wszystkie historyczne zmiany lub też zobaczyć jakie dokładnie operacje zostały wykonane na repozytorium.

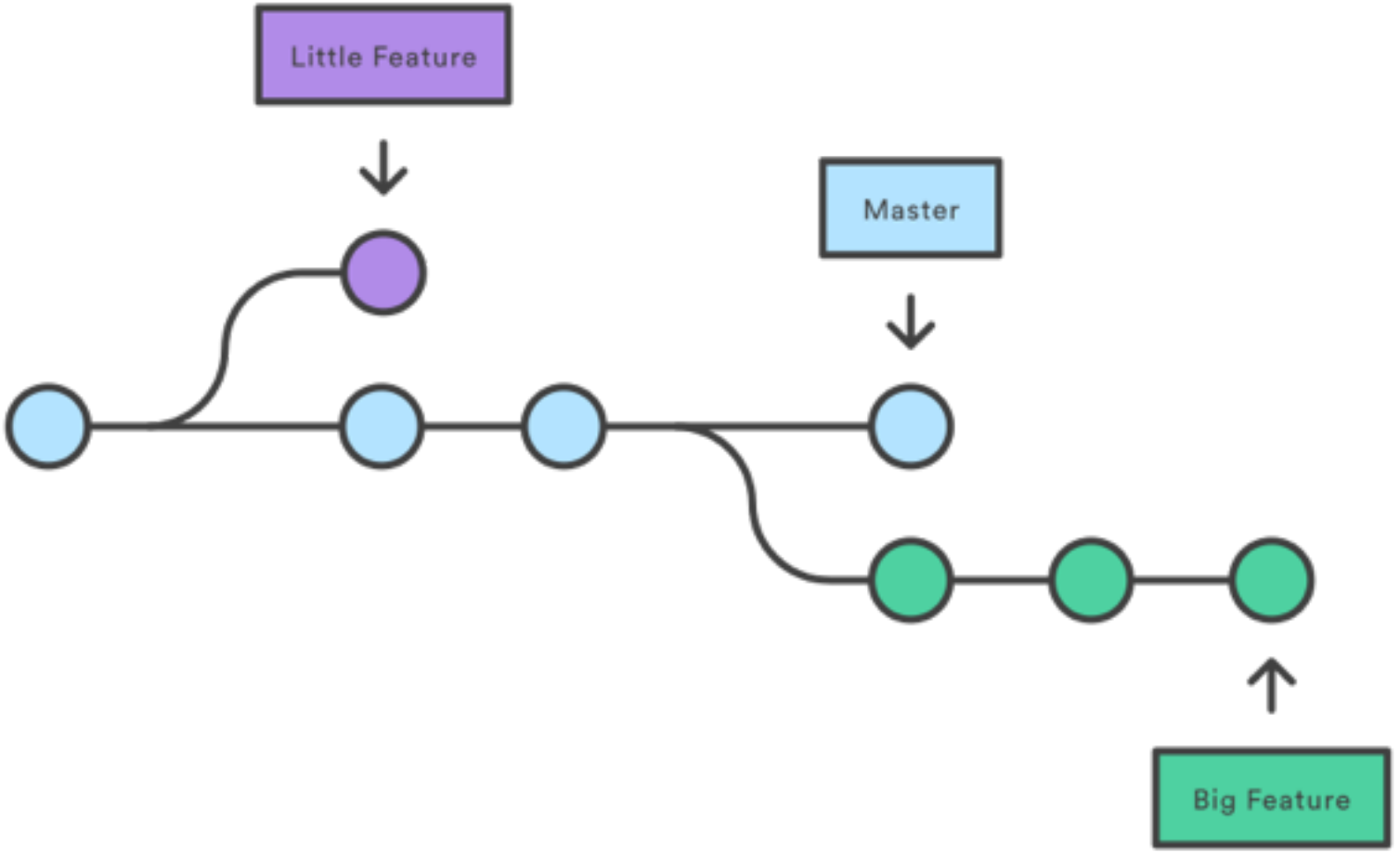
GIT - jak działa

3 stany GIT'a



GIT - jak działa

Git pozwala na branchowanie



GIT - podstawowe operacje

Zanim zaczniemy pracę z GIT

`git config --global user.name "[name]"`

Sets the name you want attached to your commit transactions

`git config --global user.email "[email address]"`

Sets the email you want attached to your commit transactions

GIT - podstawowe operacje

Tworzenie repozytorium

`git init [project-name]`

Creates a new local repository with the specified name

`git clone [url]`

Downloads a project and its entire version history

GIT - podstawowe operacje

Praca z GIT

`git status`
Lists all new or modified files to be committed

`git diff`
Shows file differences not yet staged

`git add [file]`
Snapshots the file in preparation for versioning

GIT - podstawowe operacje

Praca z GIT

`git diff --staged`

Shows file differences between staging and the last file version

`git reset [file]`

Unstages the file, but preserve its contents

`git commit -m "[descriptive message]"`

Records file snapshots permanently in version history

GIT - podstawowe operacje

GIT reset vs revert

`git revert [commit]`

Generate a new commit that undoes all of the changes introduced in <commit>, then apply it to the current branch

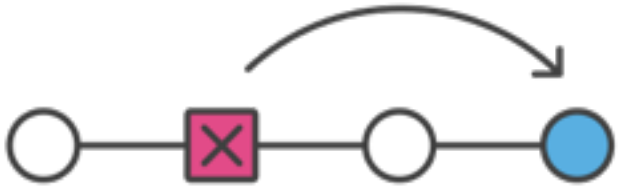
`git reset –hard [commit]`

Move the current branch tip backward to <commit> and reset both the staging area and the working directory to match

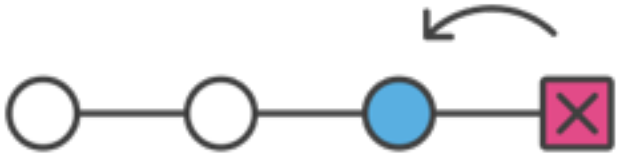
GIT - podstawowe operacje

GIT reset vs revert

Reverting

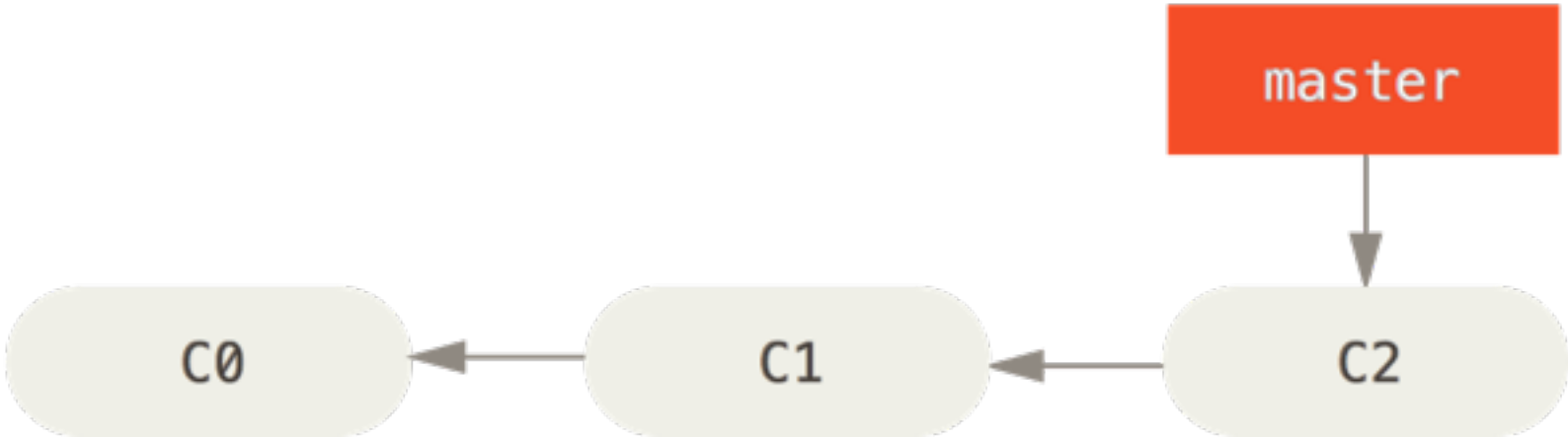


Resetting



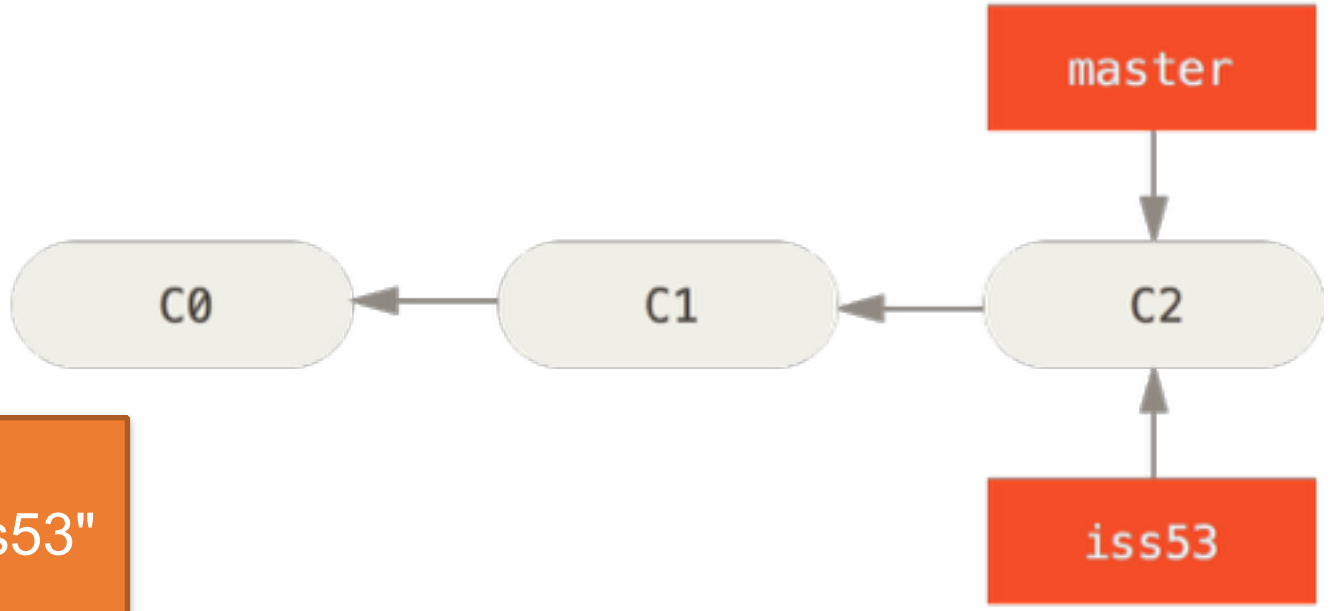
GIT - podstawowe operacje

GIT branch



GIT - podstawowe operacje

GIT branch



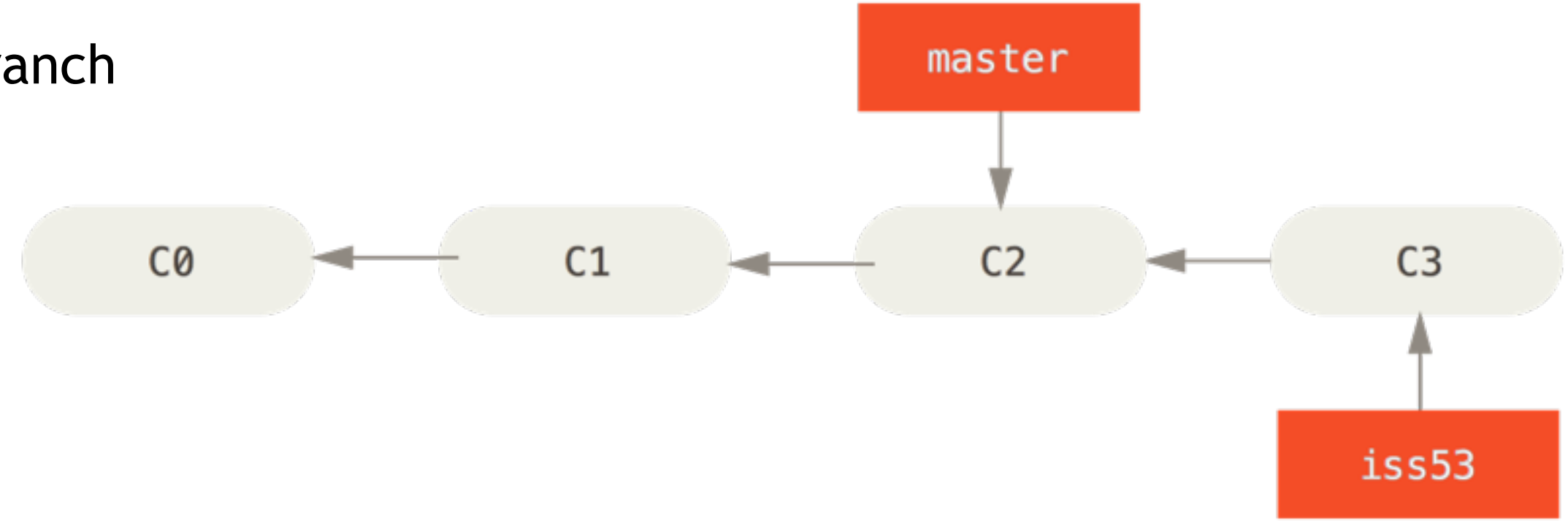

```
$ git checkout -b iss53
Switched to a new branch „iss53"

albo

$ git branch iss53
$ git checkout iss53
```

GIT - podstawowe operacje

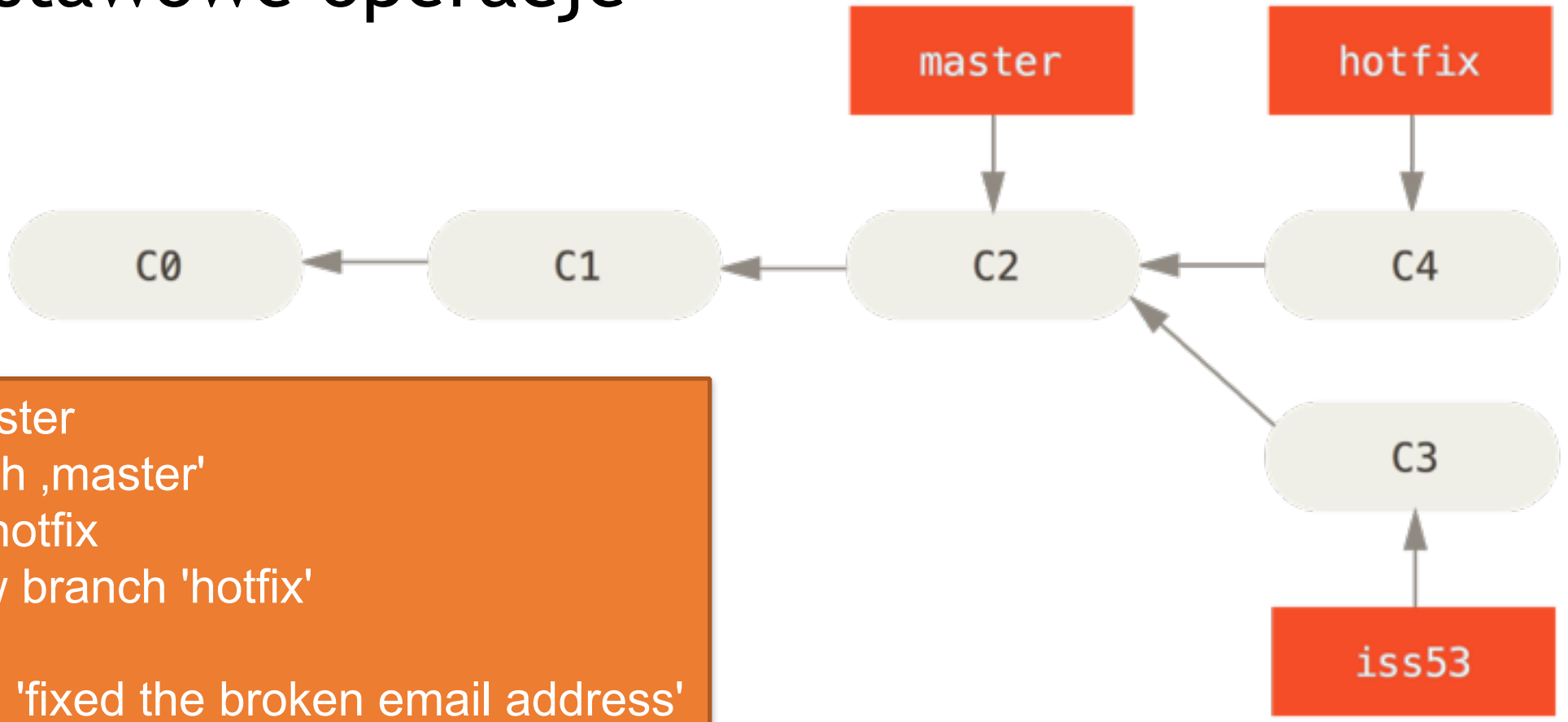
GIT branch




```
$ vim index.html
$ git commit -a -m 'added a new footer [issue 53]'
```

GIT - podstawowe operacje

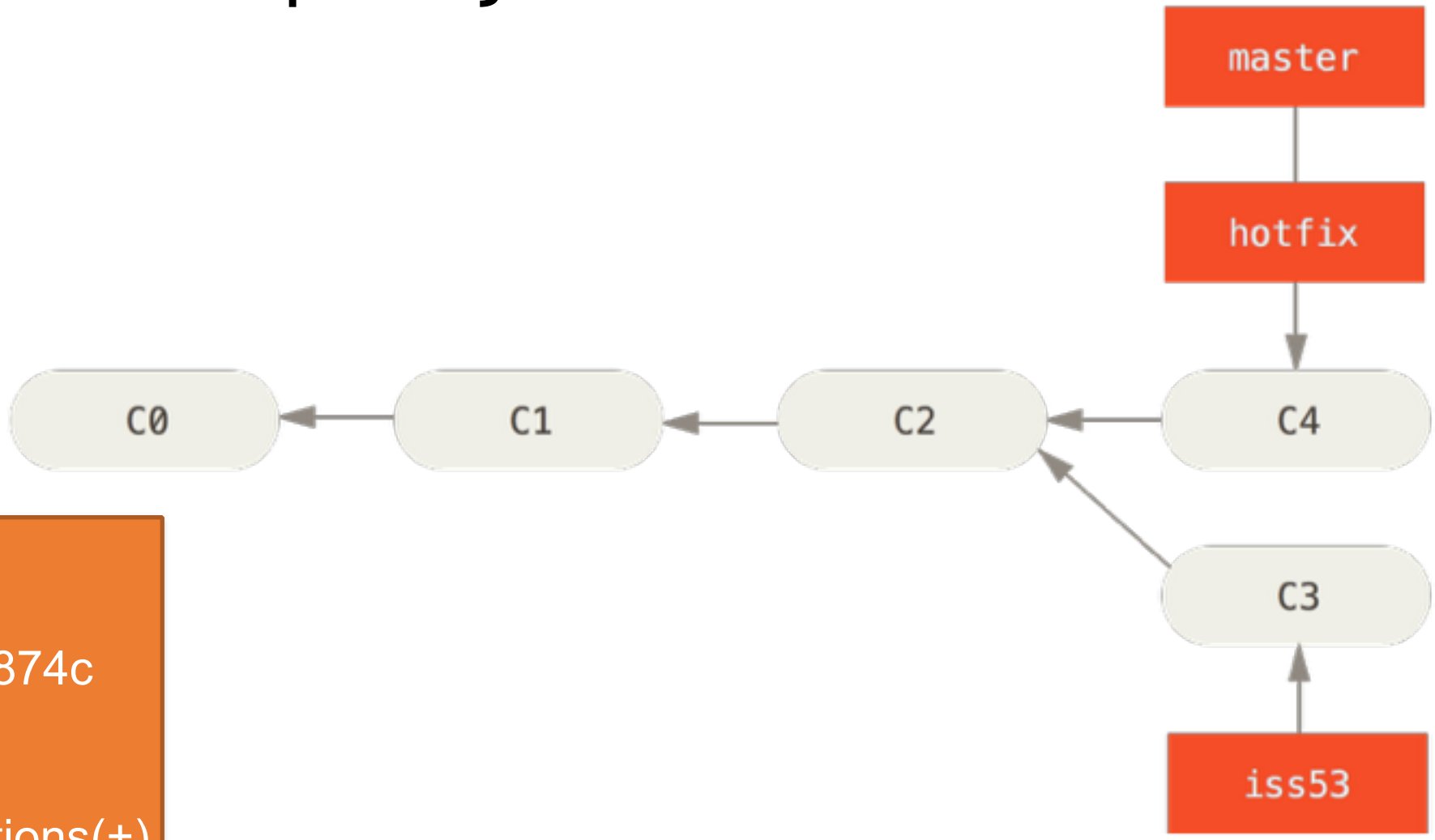
GIT branch




```
$ git checkout master
Switched to branch ‚master'
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
$ vim index.html
$ git commit -a -m 'fixed the broken email address'
[hotfix 1fb7853] fixed the broken email address
 1 file changed, 2 insertions(+)
```

GIT - podstawowe operacje

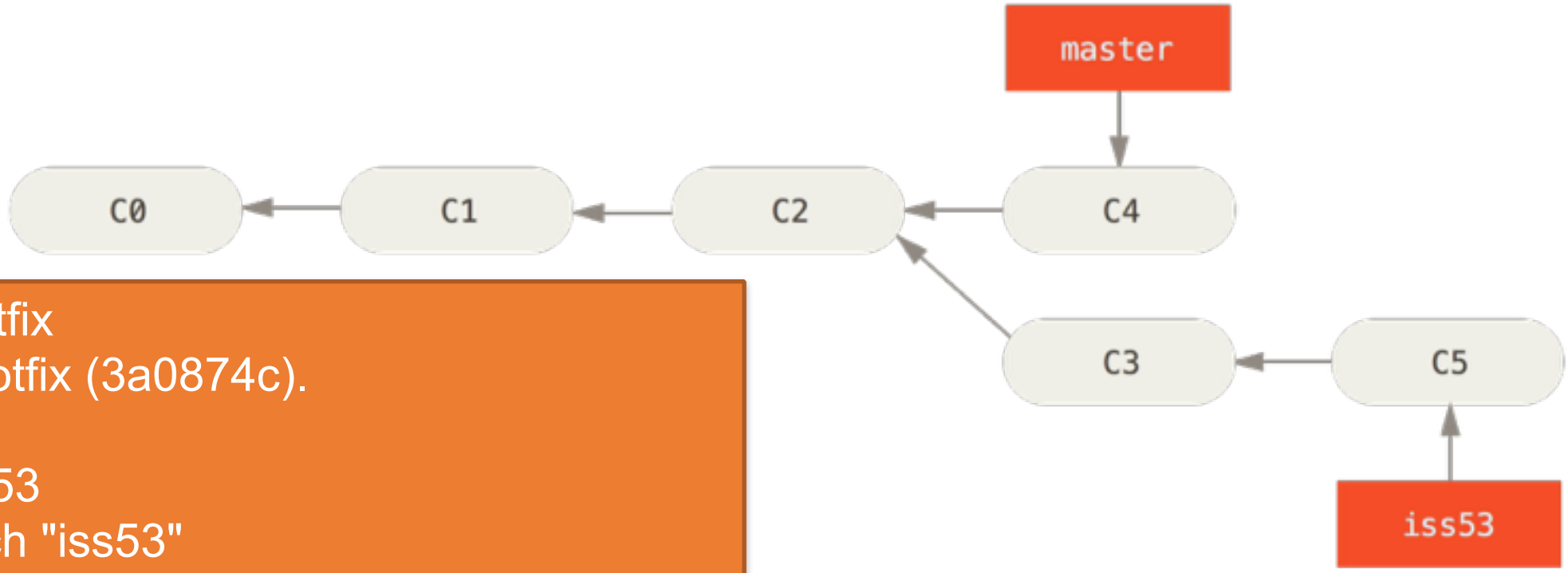
GIT branch




```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```

GIT - podstawowe operacje

GIT branch

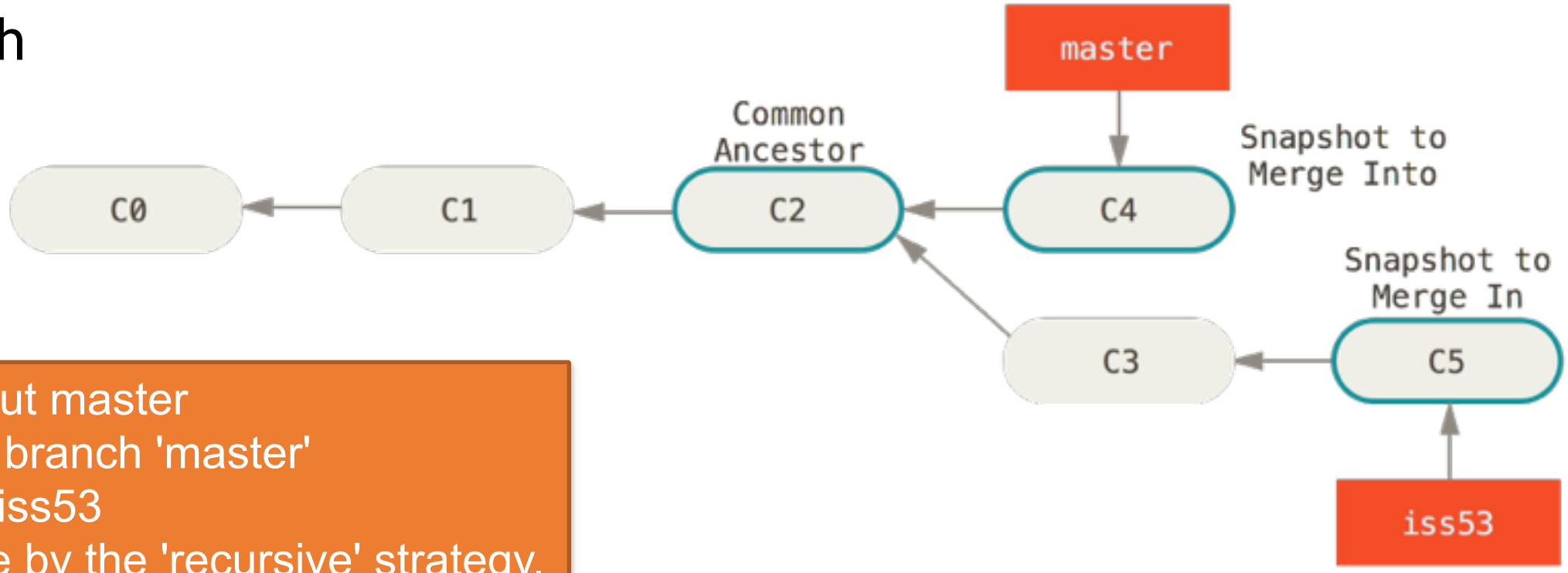



```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).

$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'finished the new footer [issue 53]'
[iss53 ad82d7a] finished the new footer [issue 53]
1 file changed, 1 insertion(+)
```

GIT - podstawowe operacje

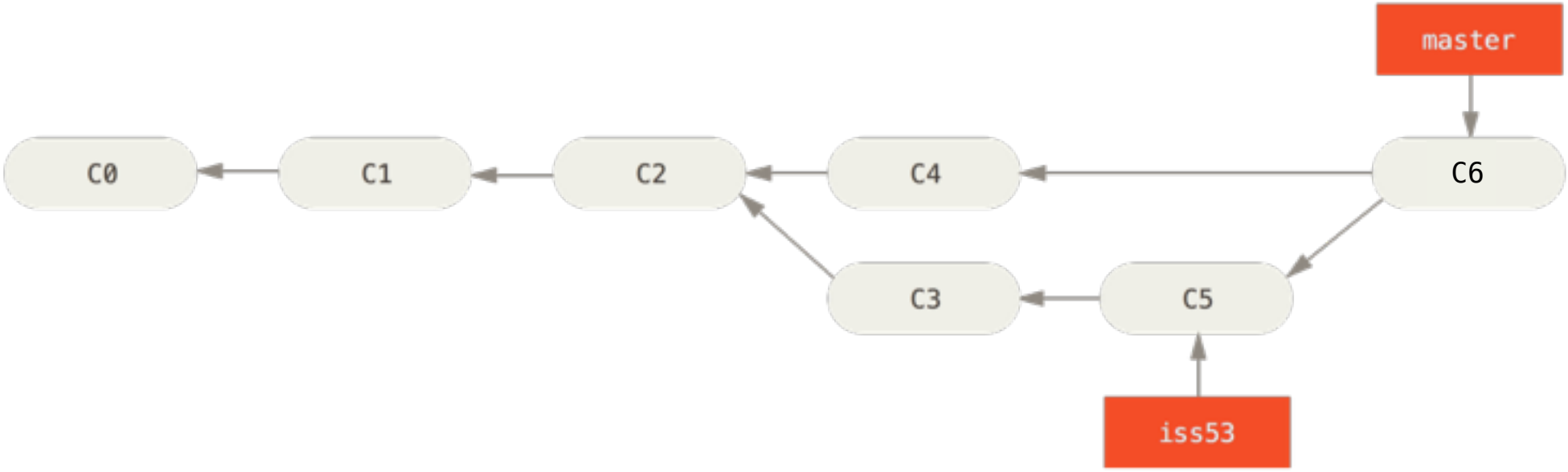
GIT branch




```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```

GIT - podstawowe operacje

GIT branch




```
$ git branch -d iss53
```

GIT - podstawowe operacje

GIT branch

`git branch`

Lists all local branches in the current repository

`git branch [branch-name]`

Creates a new branch

`git checkout [branch-name]`

Switches to the specified branch and updates the working directory

GIT - podstawowe operacje

GIT branch

`git merge [branch]`

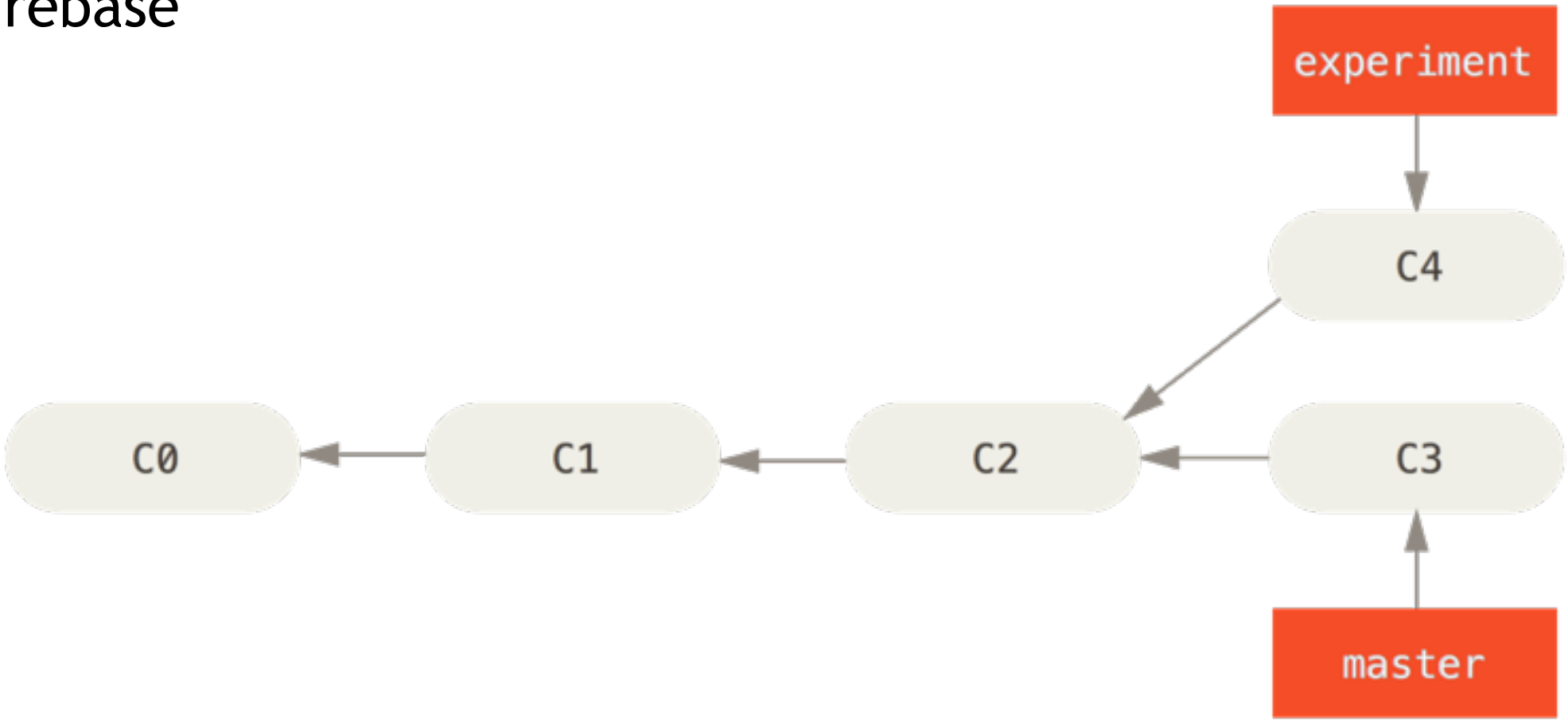
Combines the specified branch's history into the current branch

`git branch –d [branch–name]`

Deletes the specified branch

GIT - podstawowe operacje

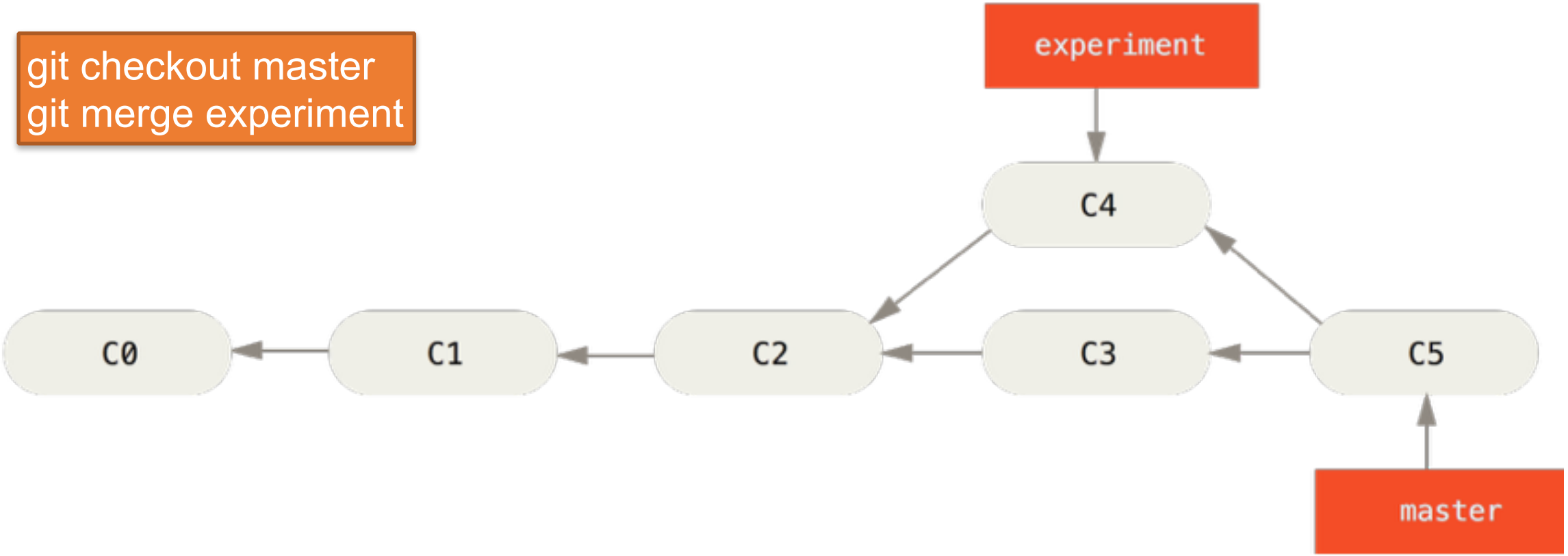
GIT rebase



GIT - podstawowe operacje

GIT rebase

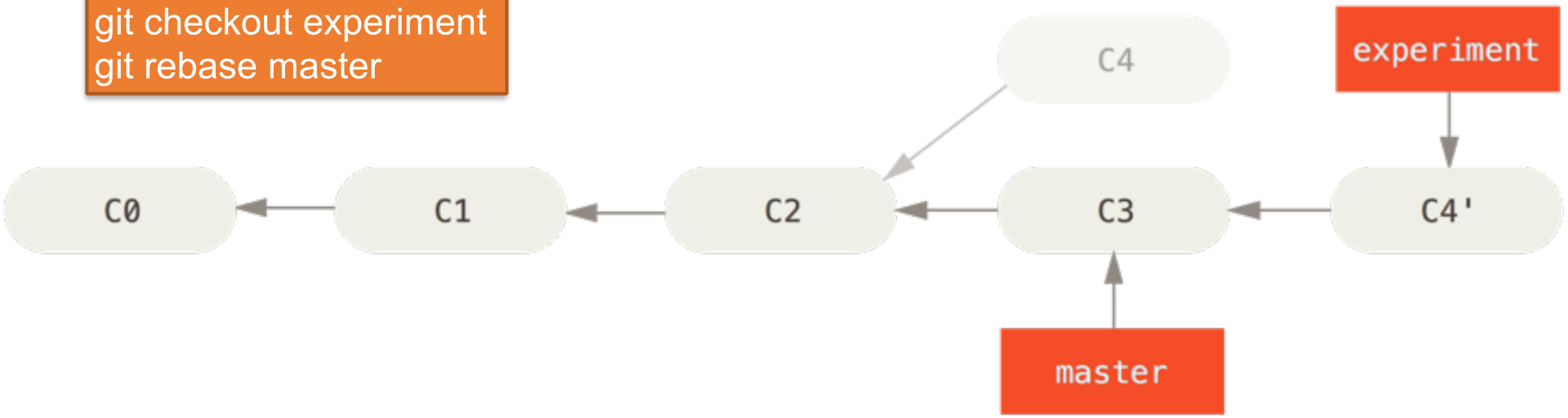
```
git checkout master
git merge experiment
```



GIT - podstawowe operacje

GIT rebase

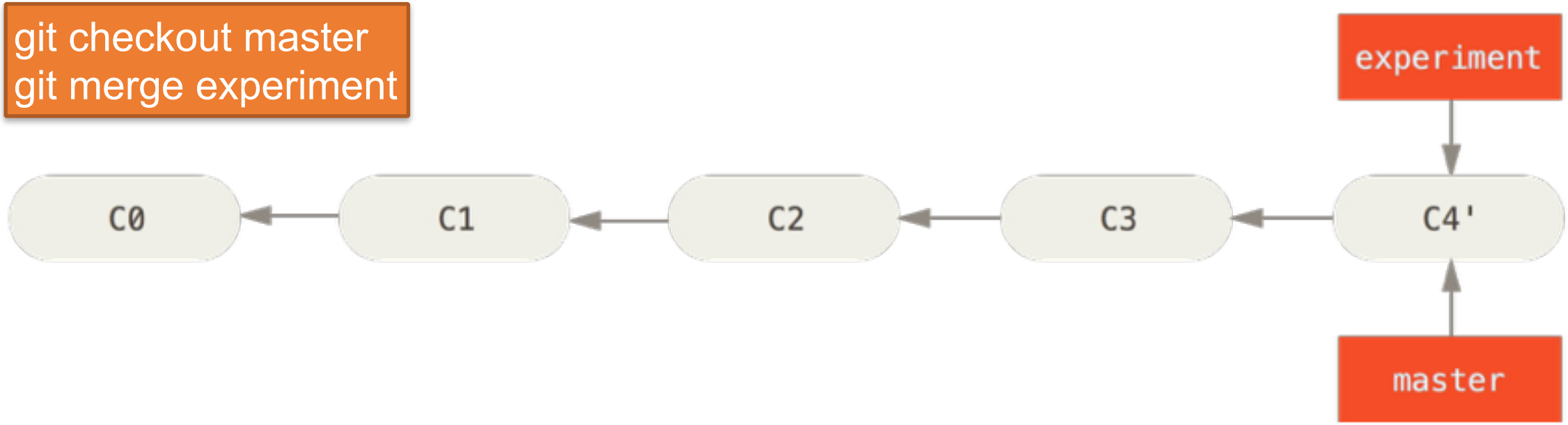
```
git checkout experiment
git rebase master
```



GIT - podstawowe operacje

GIT rebase

```
git checkout master
git merge experiment
```



GIT - podstawowe operacje

GIT praca na plikach

`git rm [file]`

Deletes the file from the working directory and stages the deletion

`git rm --cached [file]`

Removes the file from version control but preserves the file locally

`git mv [file-original] [file-renamed]`

Changes the file name and prepares it for commit

GIT - podstawowe operacje

GIT praca ze zdalnym repozytorium

`git fetch [bookmark]`

Downloads all history from the repository bookmark

`git merge [bookmark]/[branch]`

Combines bookmark's branch into current local branch

GIT - podstawowe operacje

GIT praca ze zdalnym repozytorium

`git push [alias] [branch]`

Uploads all local branch commits to GitHub

`git pull`

Downloads bookmark history and incorporates changes

GIT - podstawowe operacje

GIT tagowanie

`git tag –a [tag_name] –m [comment]`
Creates TAG *tag_name*

GIT - śledzenie historii

`git log`

Lists version history for the current branch

`git log --follow [file]`

Lists version history for a file, including renames

`git diff [first-branch]...[second-branch]`

Shows content differences between two branches

`git show [commit]`

Outputs metadata and content changes of the specified commit

GIT - śledzenie historii

```
$ git show 1c002dd4b536e7479fe34593e72e6c6c1819e53b
$ git show 1c002dd4b536e7479f
$ git show 1c002d
```

GIT - gitignore

```
.svn
deb/*.deb
src/**
*.iml
.idea
*~
```

GIT a WEBStorm

GIT Good Practice

Co i kiedy commitować

- commitujemy często*
- zmiany różnych funkcjonalności commitujemy osobno
- starajmy się unikać commitowania dużych plików, szczególnie wynikowych - w GIT trzymajmy tylko kod źródłowy.
  - jar
  - deb

Do głównej gałęzi commitujemy tylko skończone funkcjonalności!

GIT Good Practice

Nazewnictwo komentarzy

Krótki (50 znaków lub mniej) opis zmiany

Bardziej szczegółowy tekst jeżeli jest taka konieczność. Zawijaj wiersze po około 72 znakach. Czasami pierwsza linia jest traktowana jako temat wiadomości email, a reszta komentarza jako treść. Pusta linia oddzielająca opis od streszczenia jest konieczna (chyba że ominiesz szczegółowy opis kompletnie); narzędzia takie jak `rebase` mogą się pogubić jeżeli nie oddzielisz ich.

Kolejne paragrafy przychodzą po pustej linii.

- wypunktowania są poprawne, również

- zazwyczaj łącznik lub gwiazdka jest używana do punktowania, poprzedzona pojedynczym znakiem spacji, z pustą linią pomiędzy, jednak zwyczaje mogą się tutaj różnić.

GIT Good Practice

Nazewnictwo komentarzy

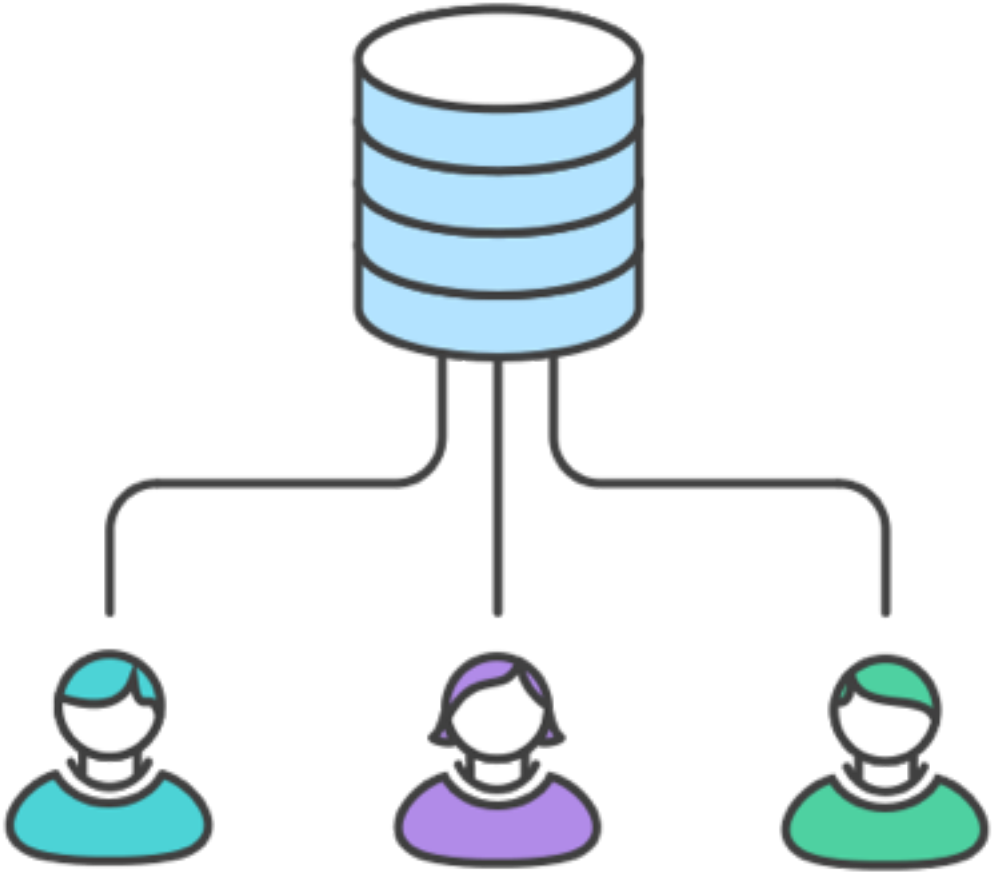
Cechy dobrych komentarzy:

- są zrozumiałe nie tylko dla autora a również dla reszty zespołu:
- komentarze nie muszą być długie - może to być np. numer ticketu z bugtrackera + krótki opis
- zespól na początku przed rozpoczęciem pracy nad projektem powinien ustalić zasady komentowania

GIT rozwiązywanie konfliktów

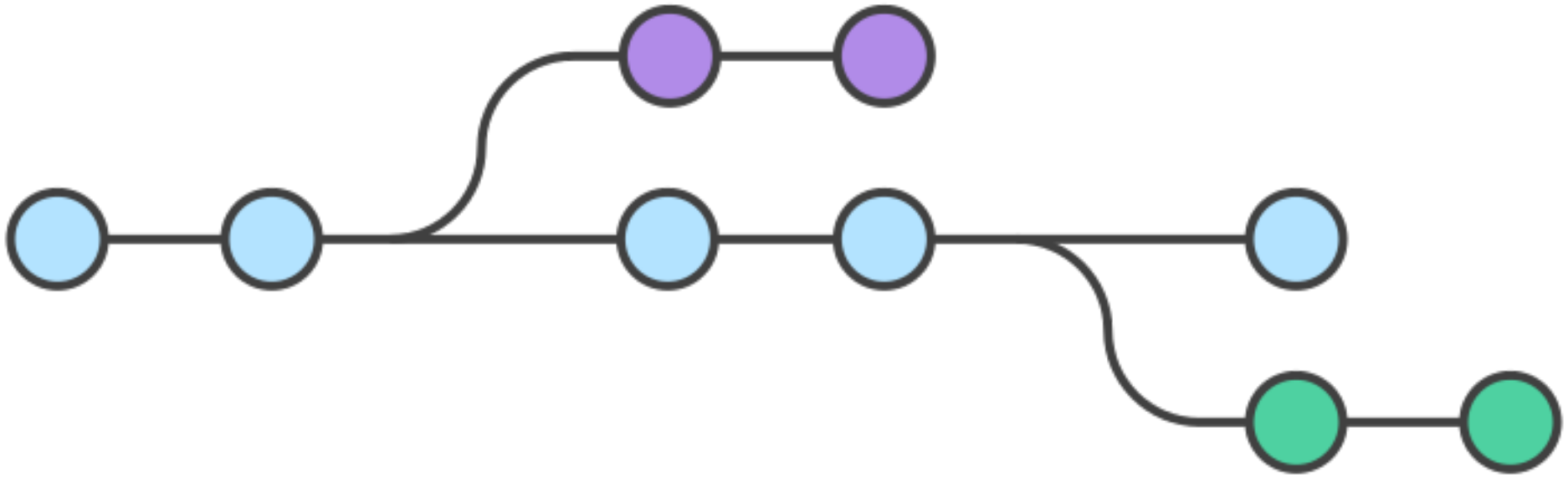
GIT cd

GIT workflow— centralized Workflow



GIT cd

GIT workflow— Feature branch



GIT cd

GIT workflow— gitflow

GIT cd

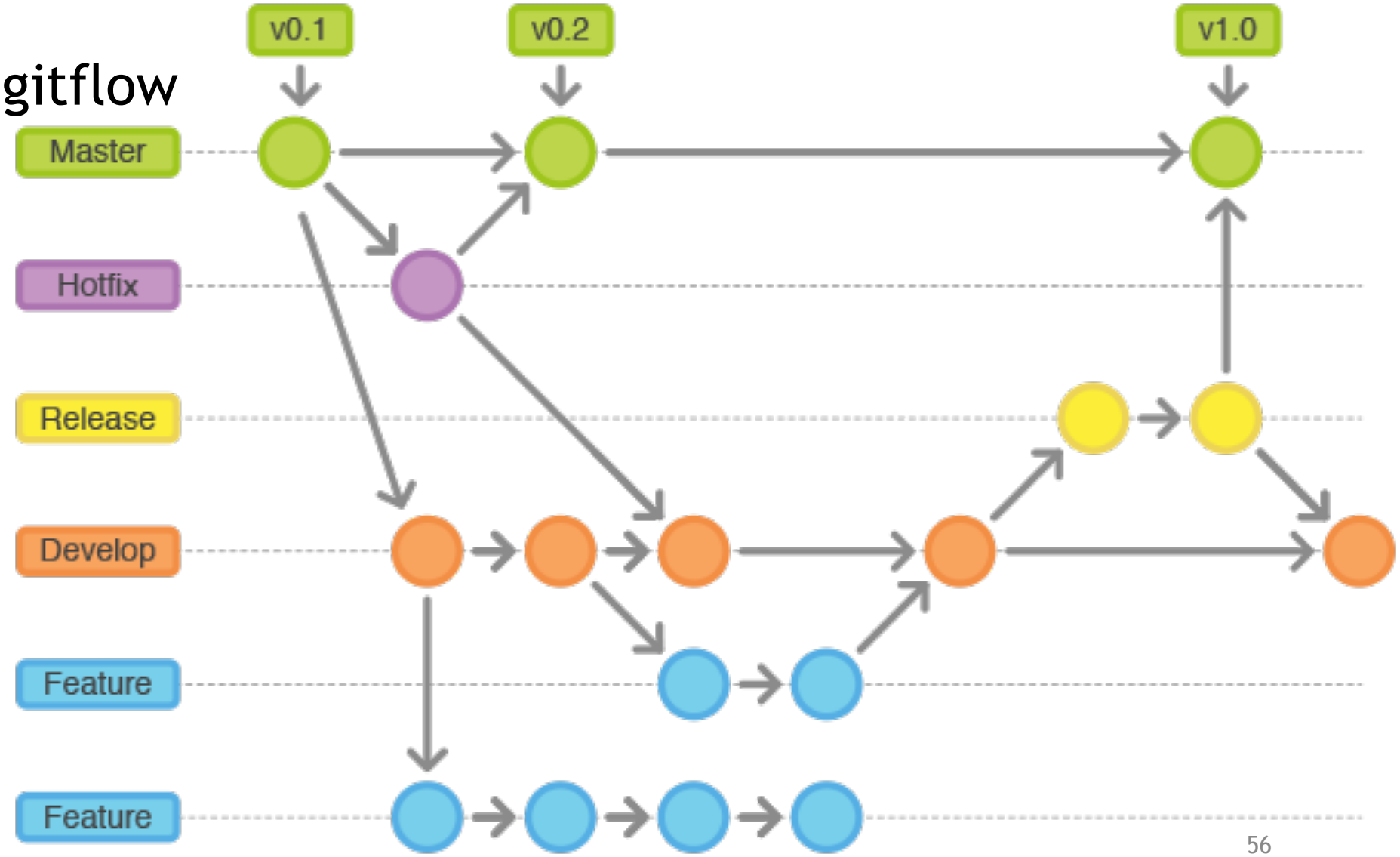
GIT workflow— gitflow

GIT cd

GIT workflow— gitflow

GIT cd

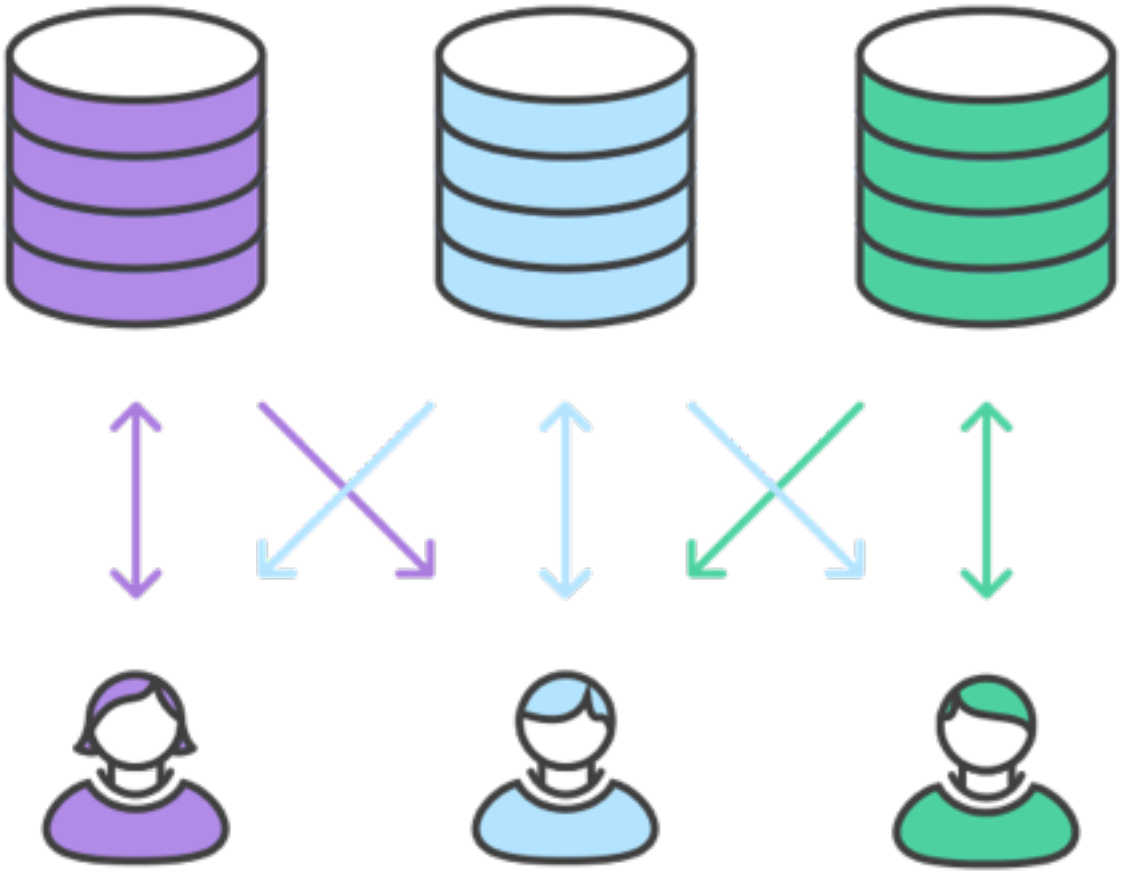
GIT workflow– gitflow



GIT cd

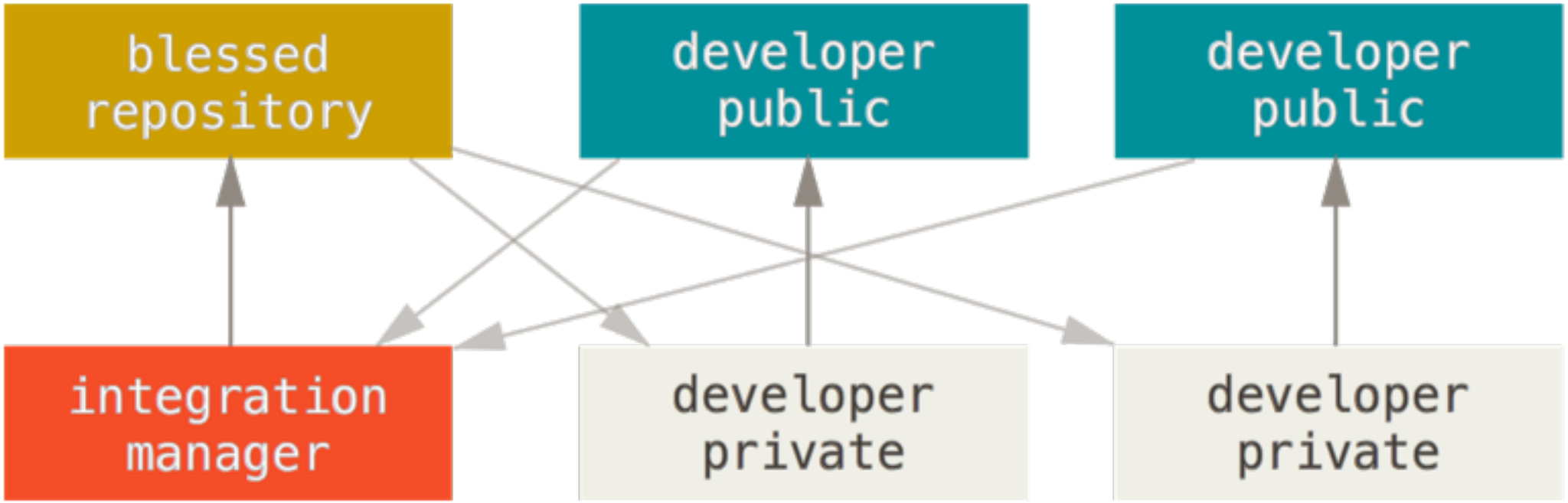
GIT cd

GIT workflow - Forking Workflow



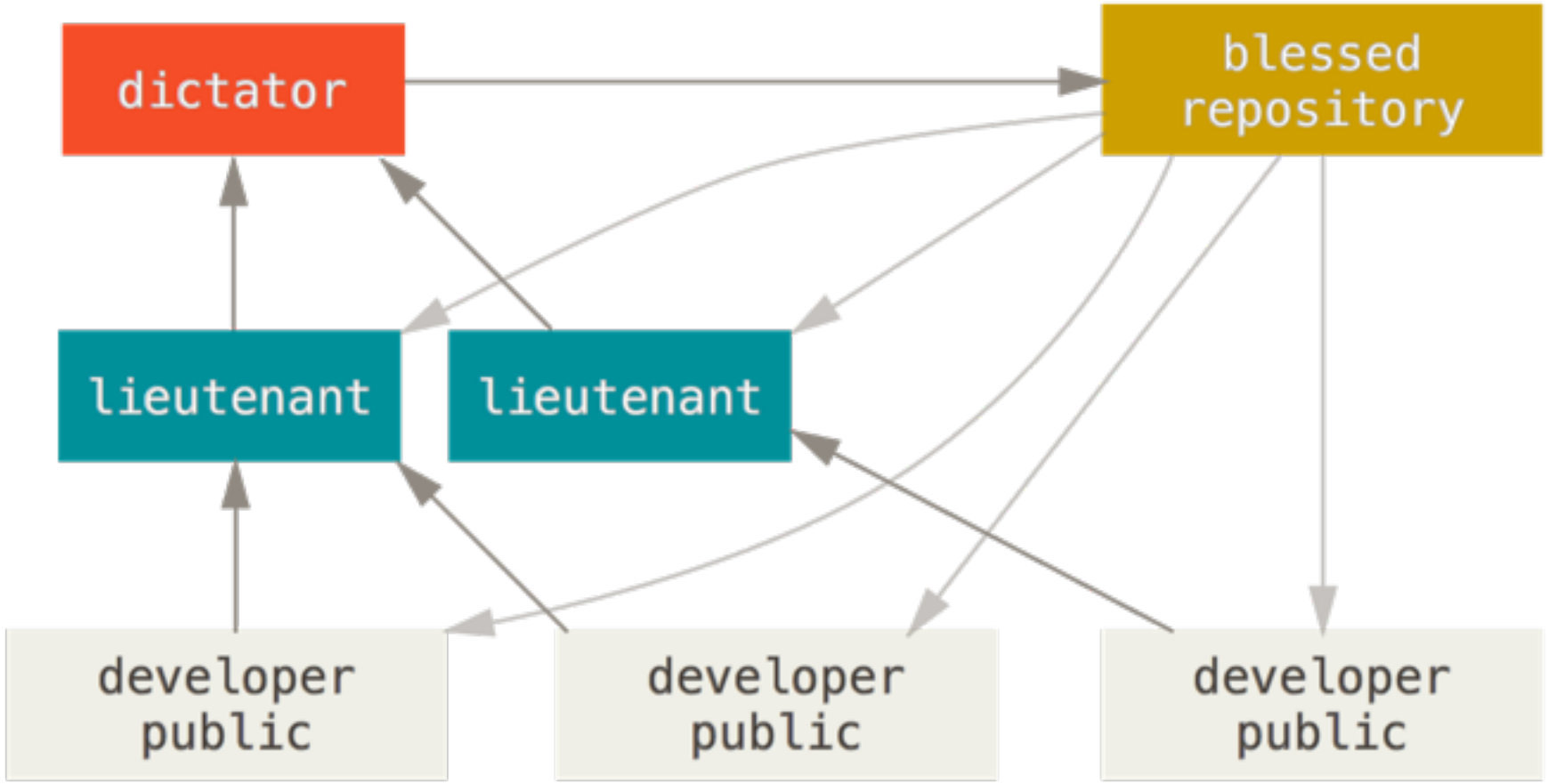
GIT cd

GIT workflow - Forking Workflow



GIT cd

GIT workflow - Forking Workflow



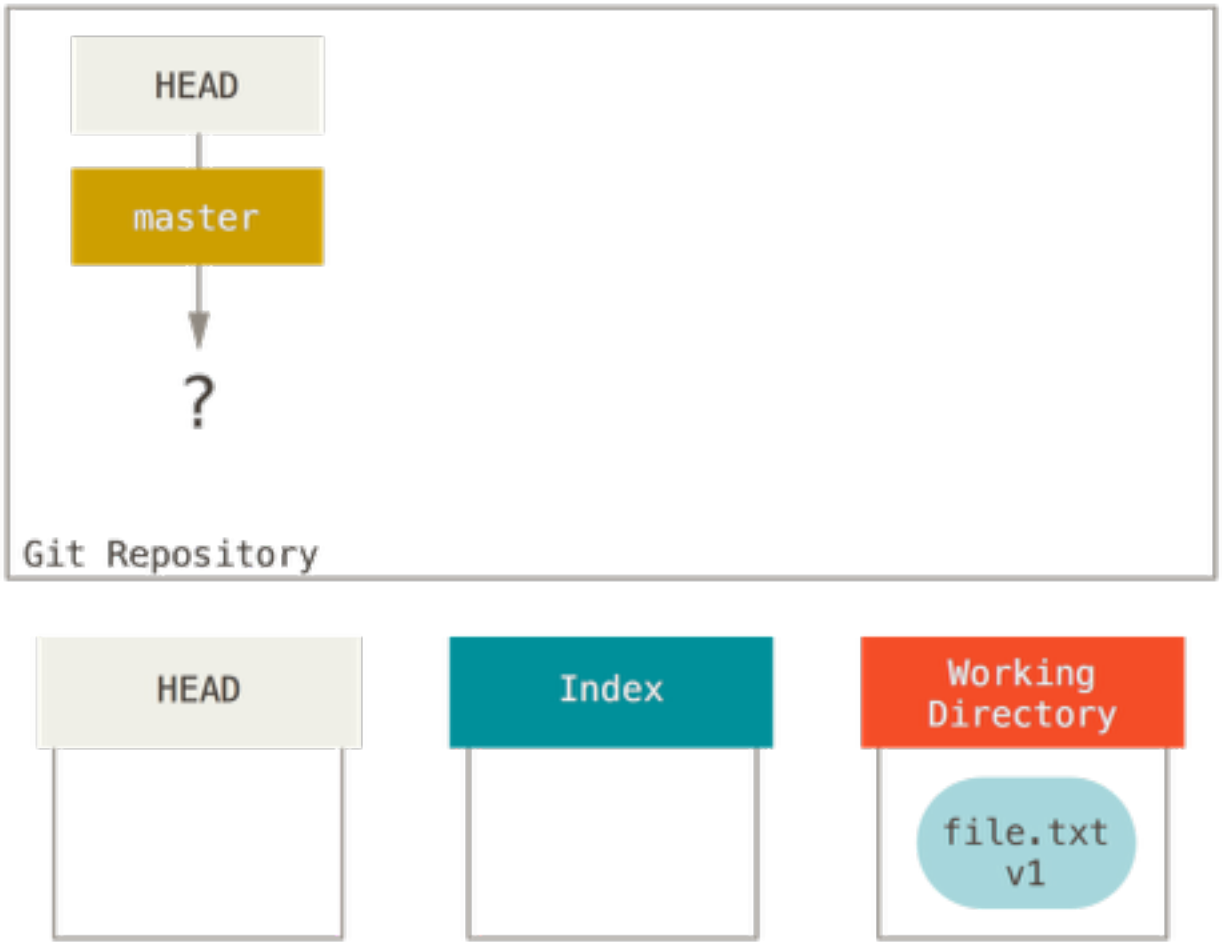
GIT pull-request

1. git commit

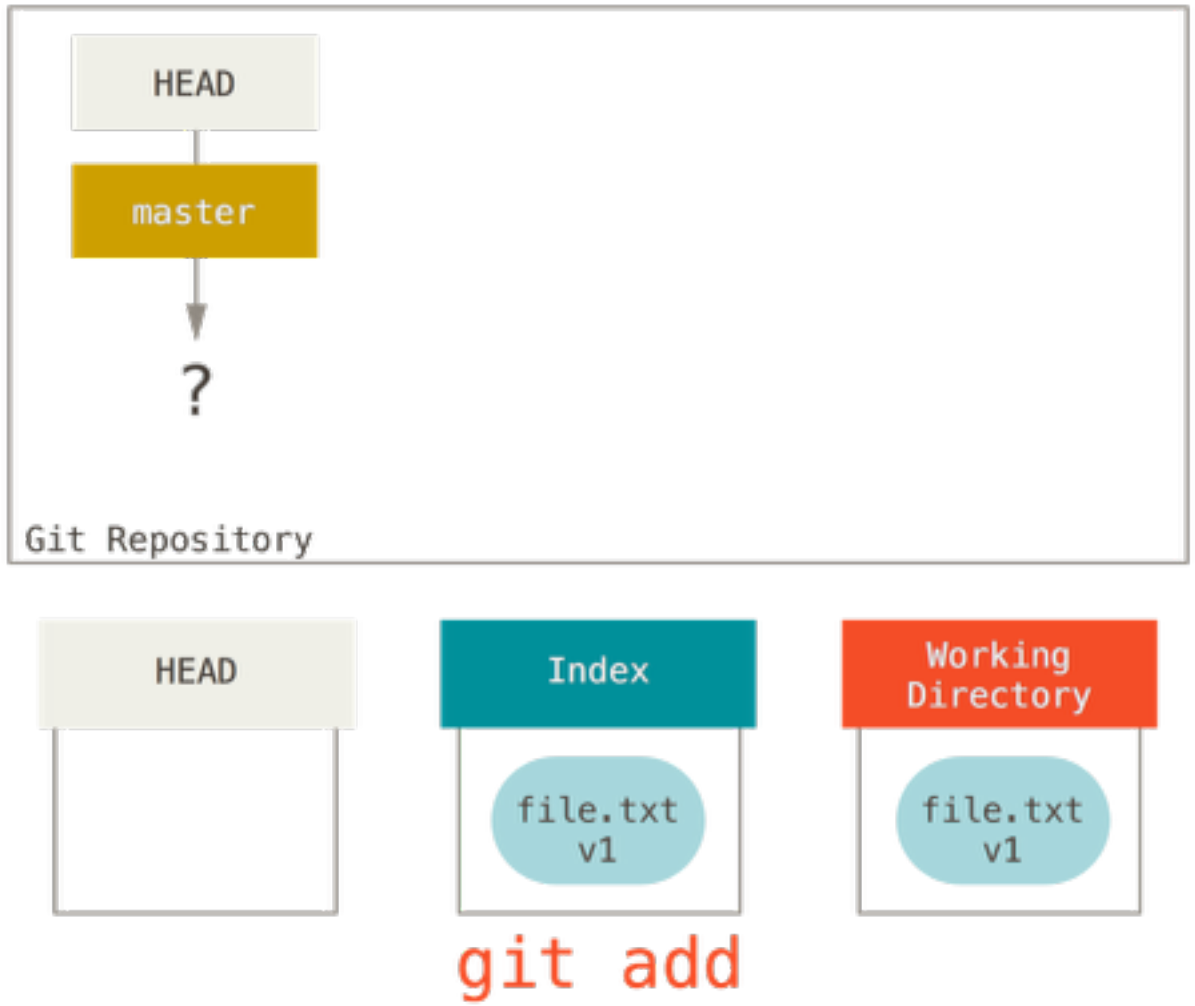
2. git push

3. leave building

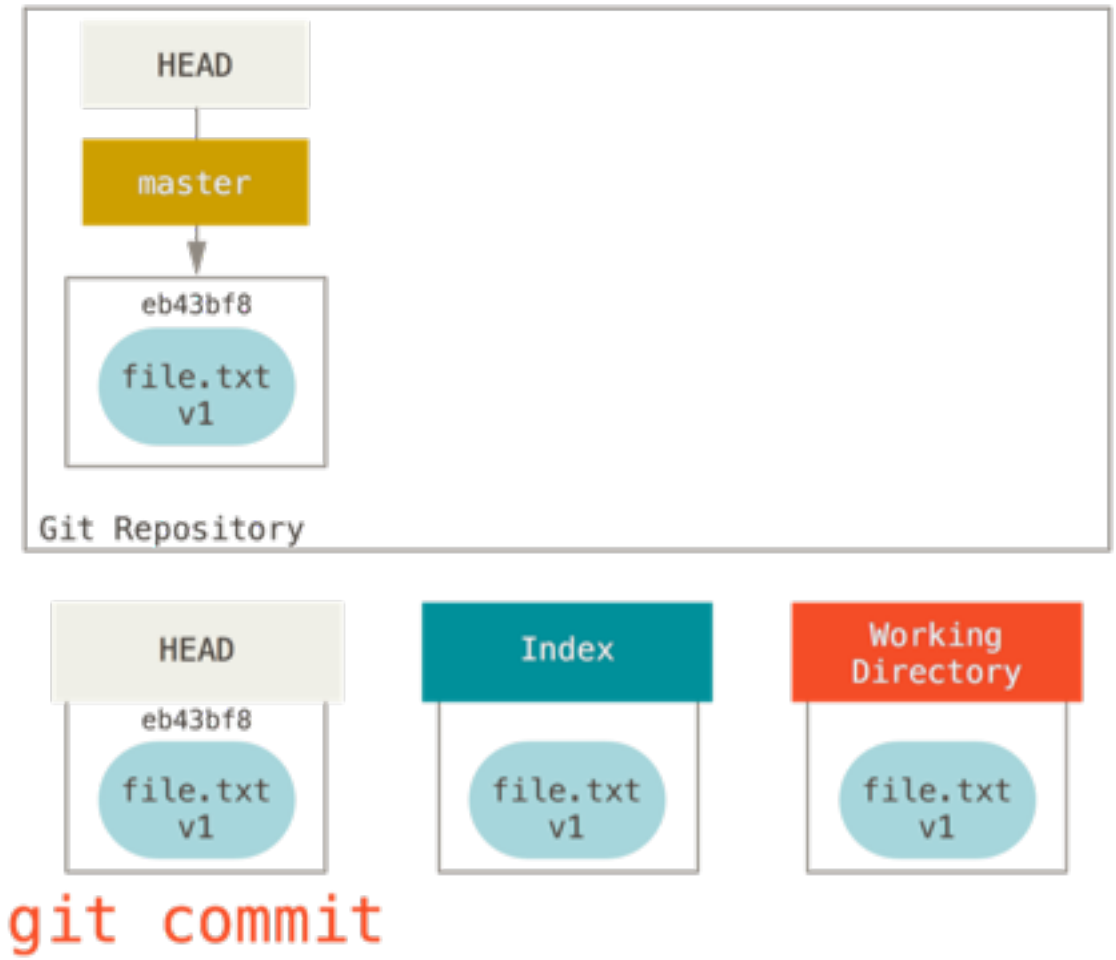
GIT reset demistified…



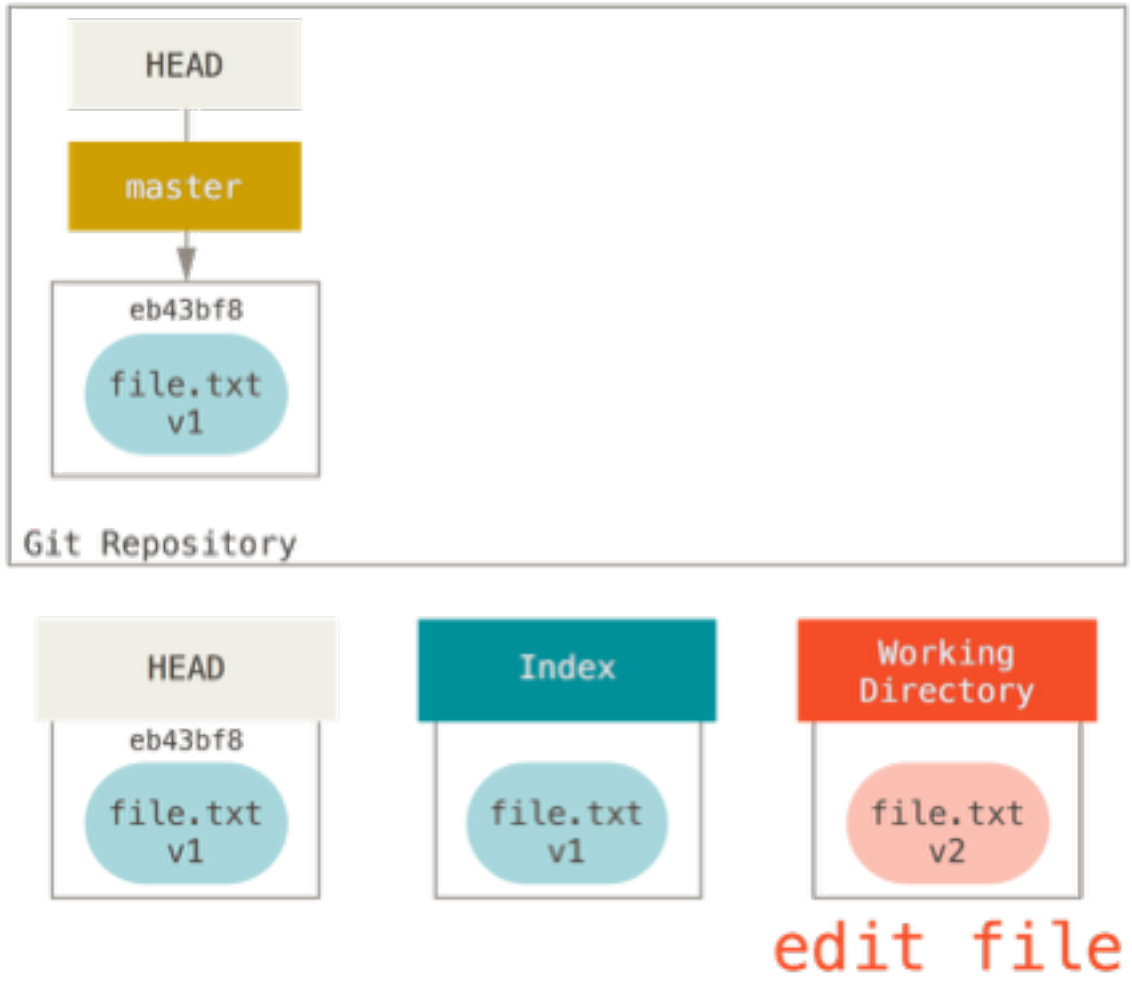
GIT reset demistified…



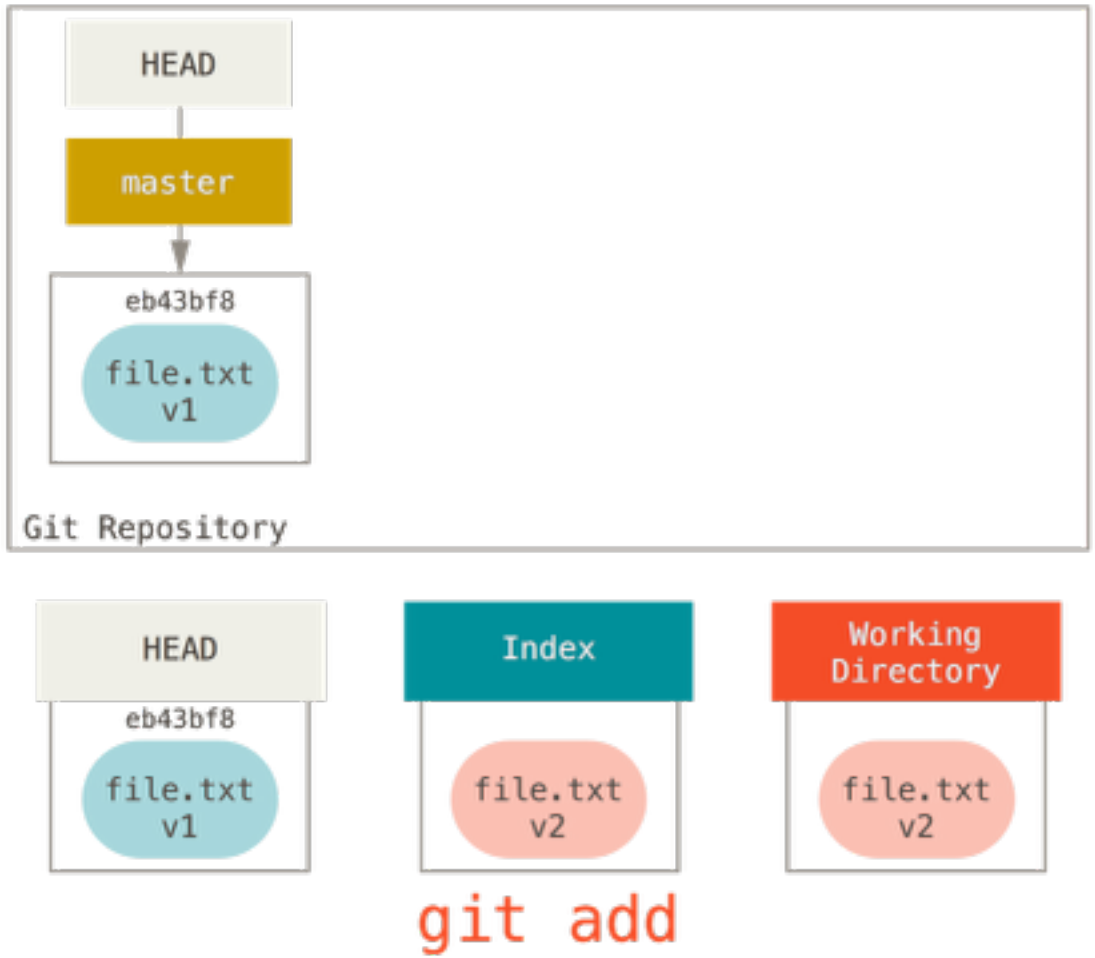
GIT reset demistified…



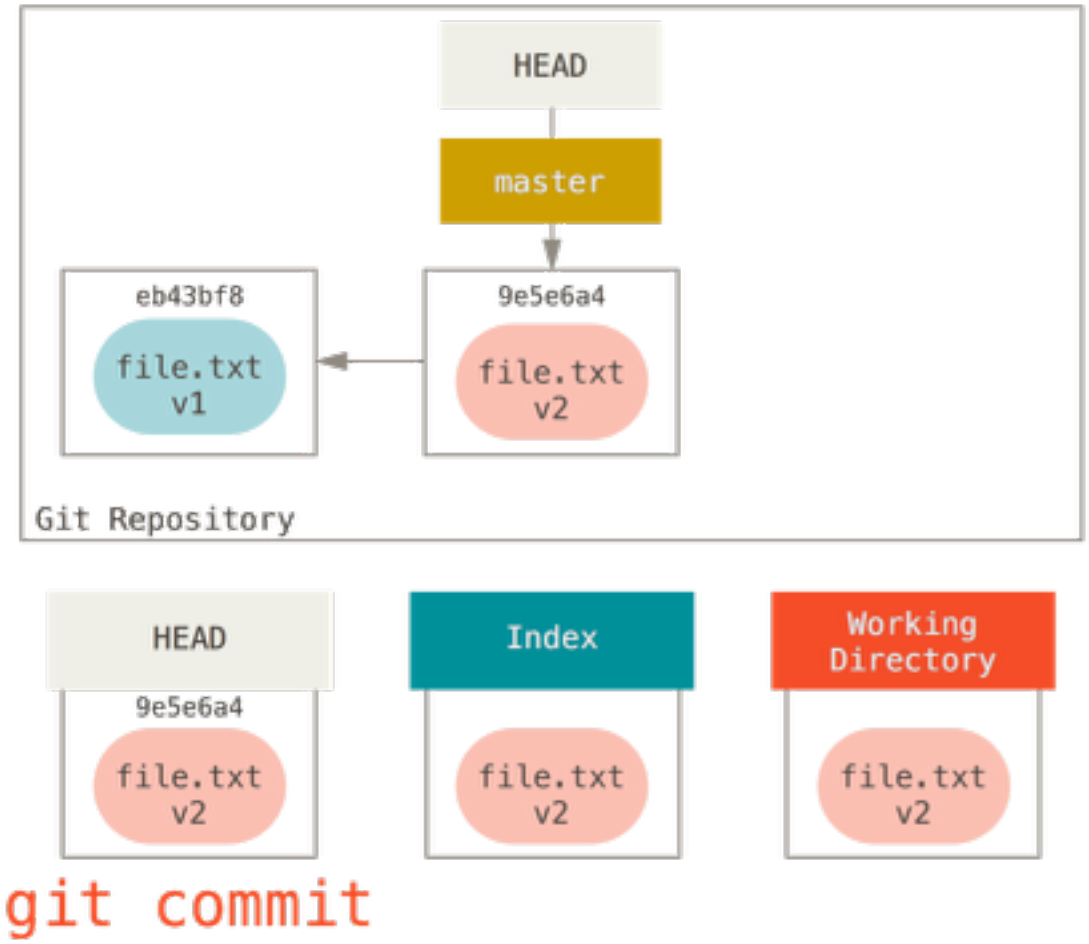
GIT reset demistified…



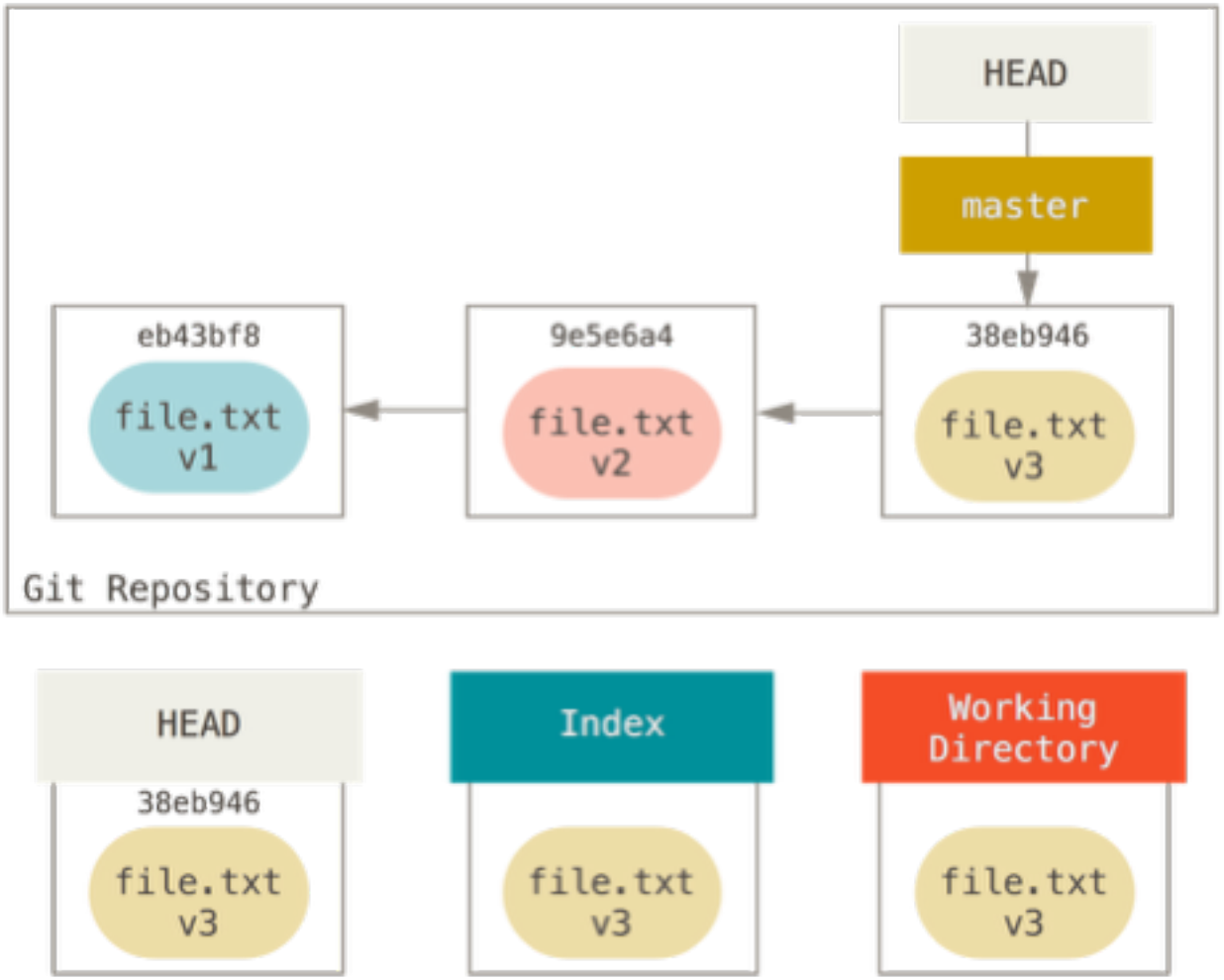
GIT reset demistified…



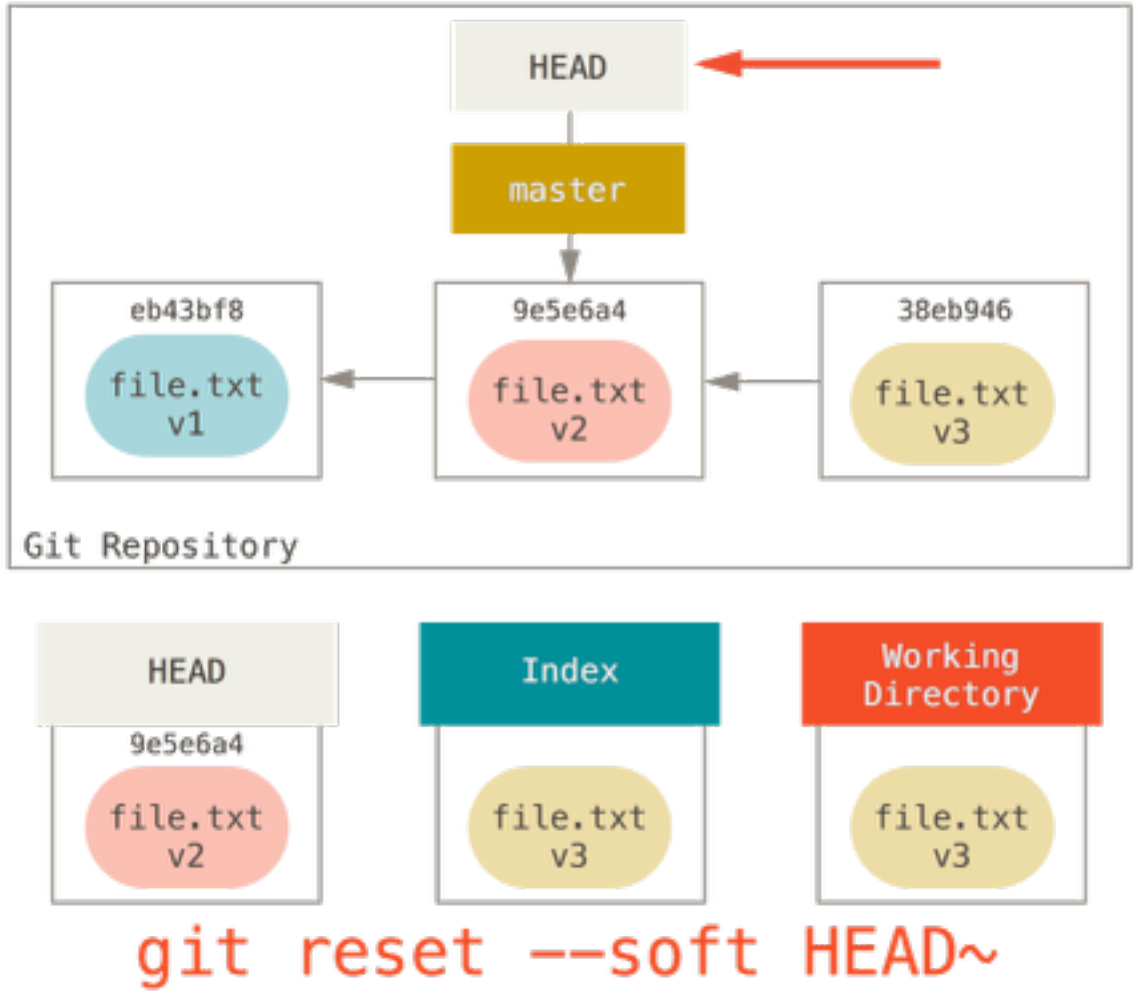
GIT reset demistified…



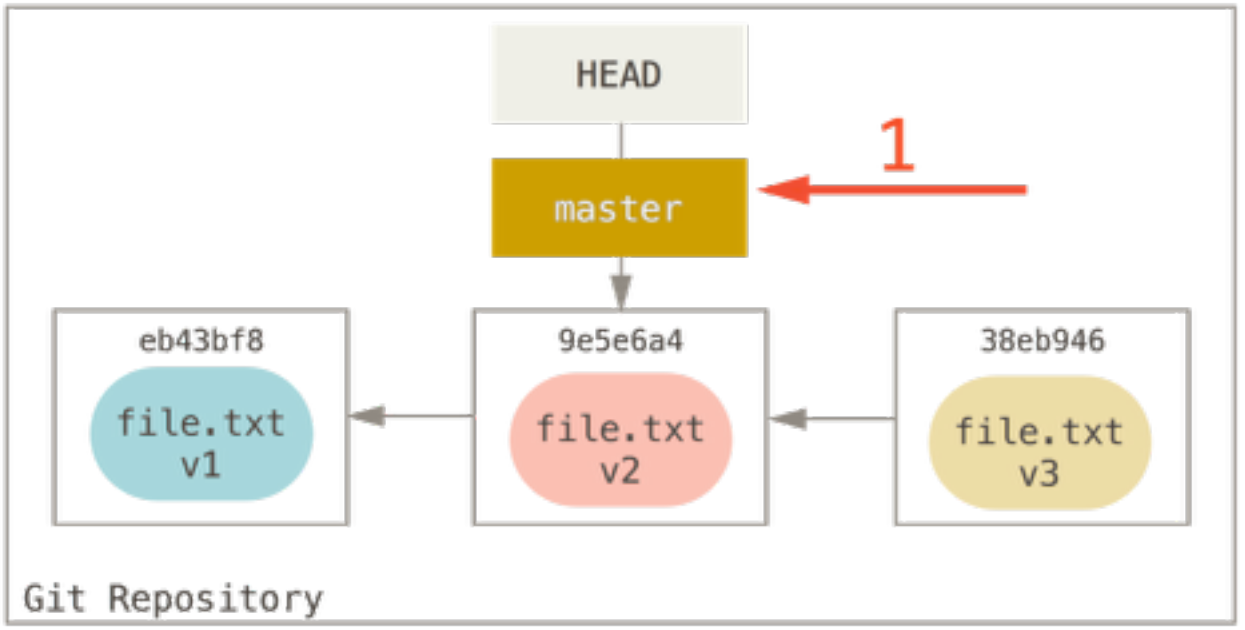
GIT reset demistified…

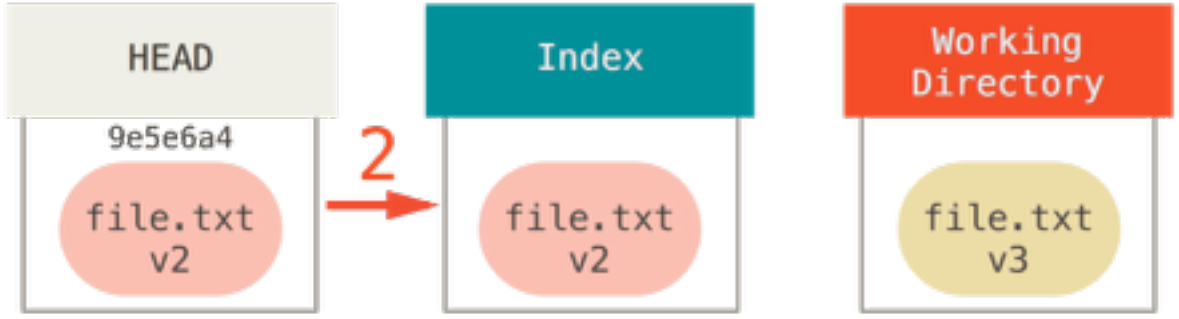


GIT reset demistified…



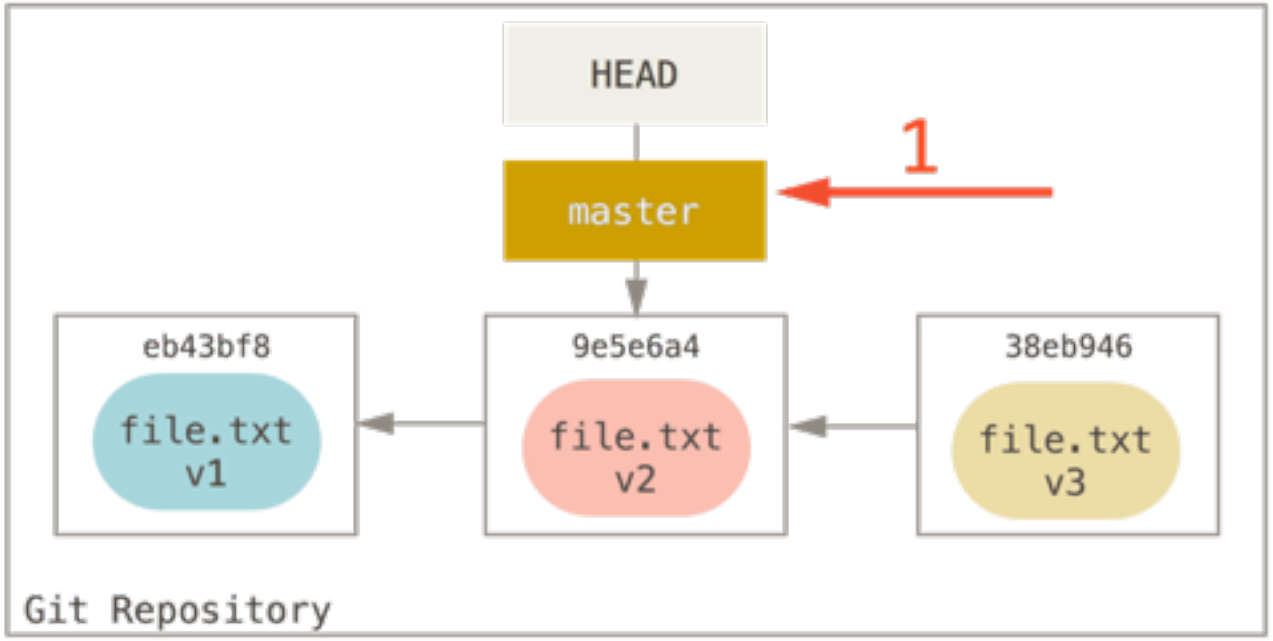
GIT reset demistified…

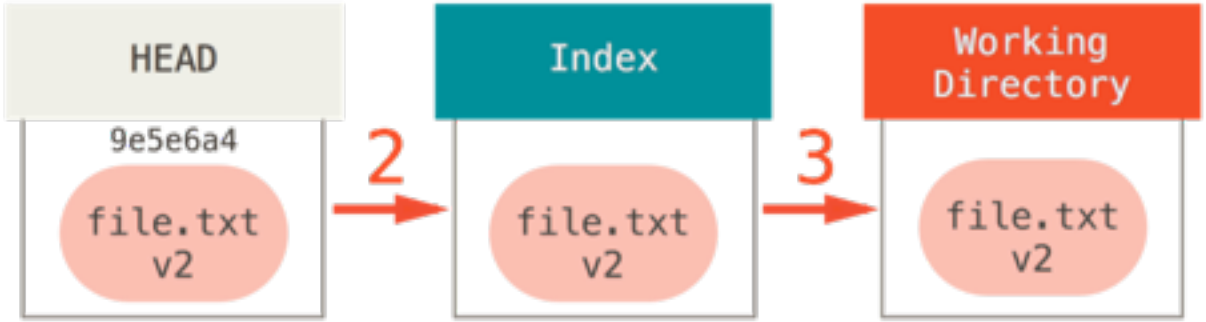




git reset [--mixed] HEAD~

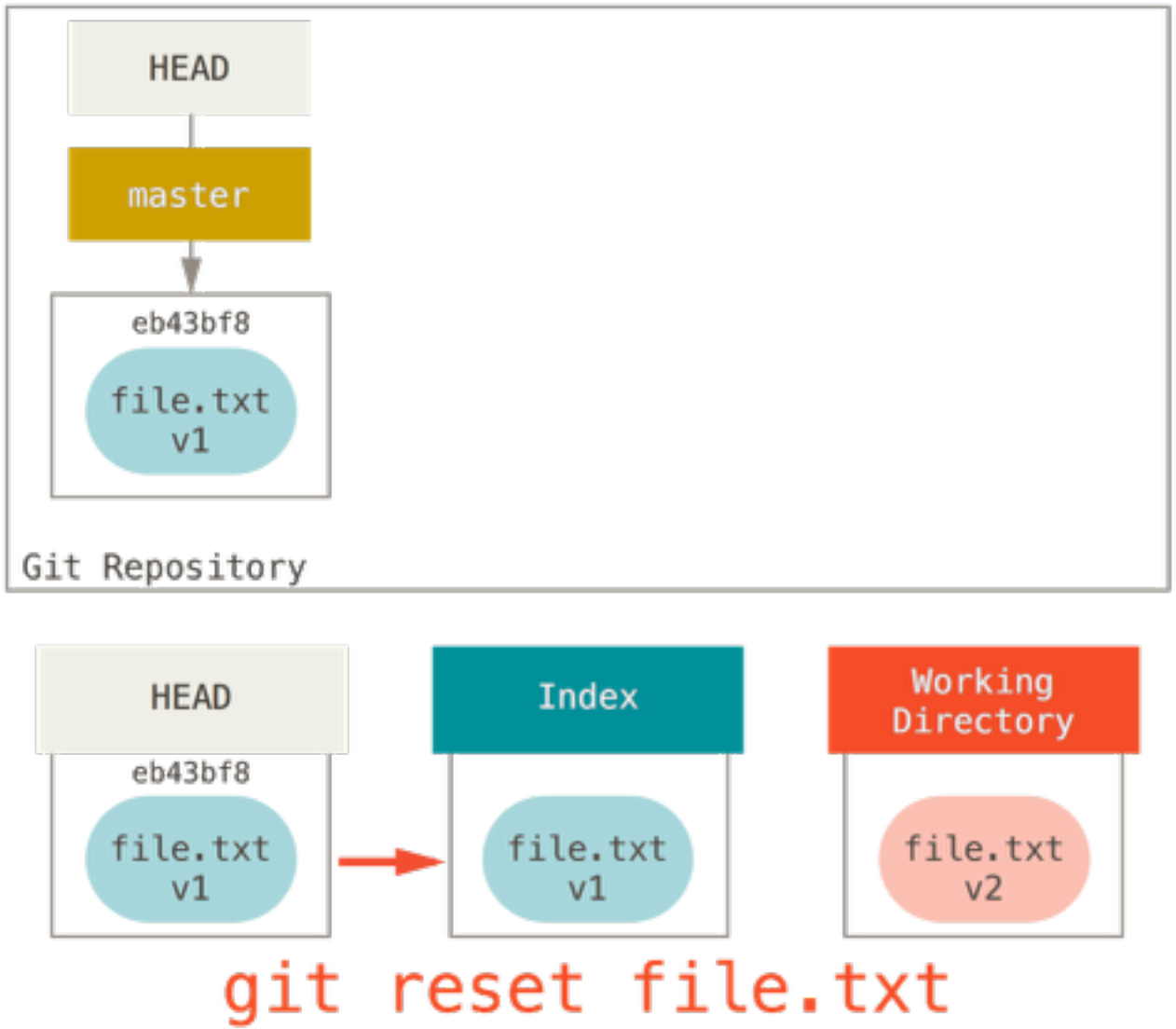
GIT reset demistified…






```
git reset --hard HEAD~
```

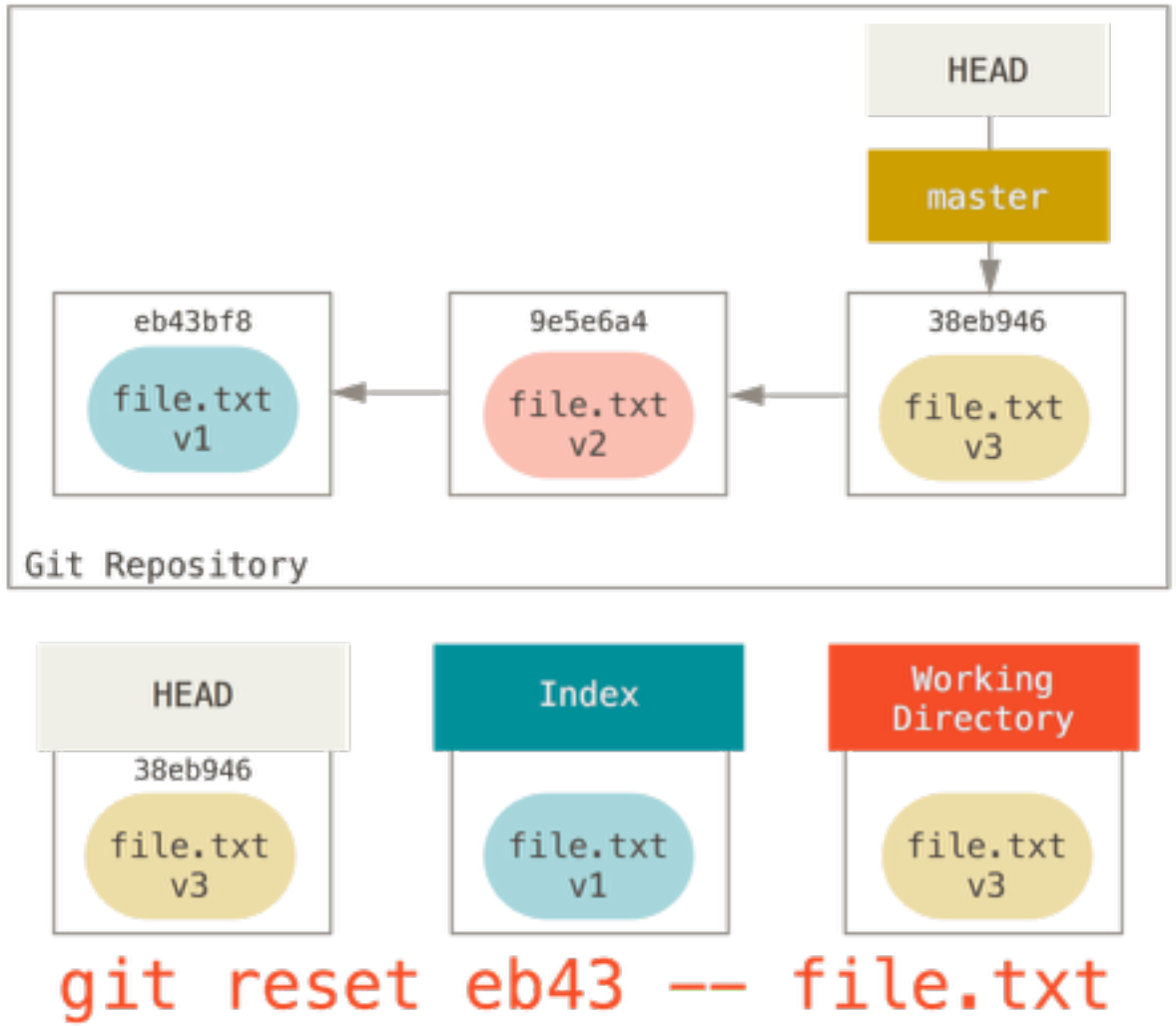
GIT reset demistified…



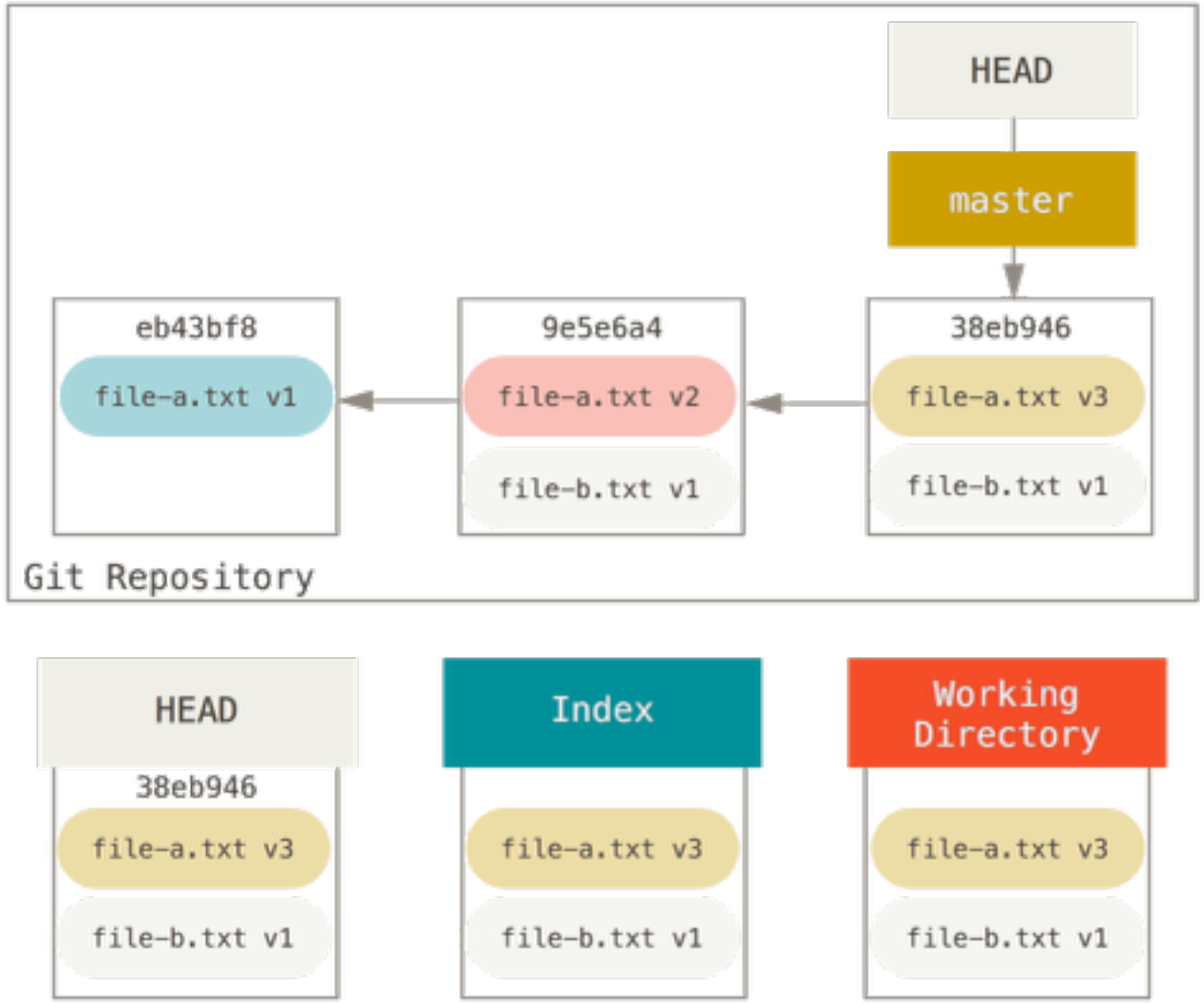
GIT reset demistified…

git add file.txt

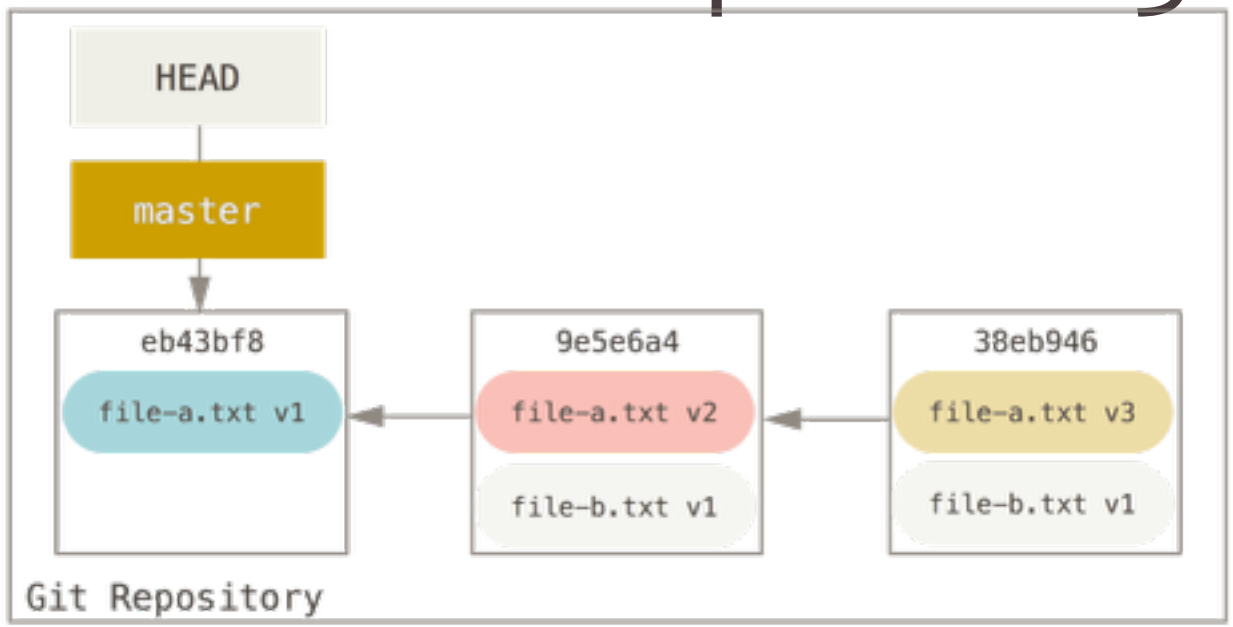
GIT reset demistified…

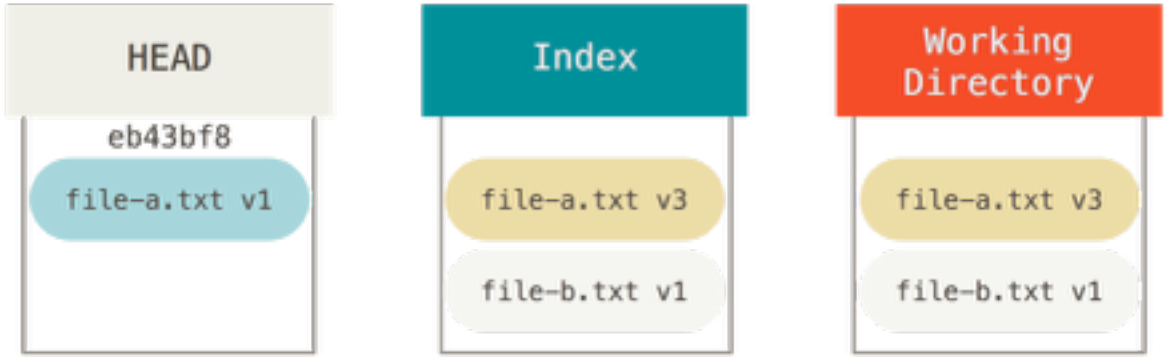


GIT reset squashing



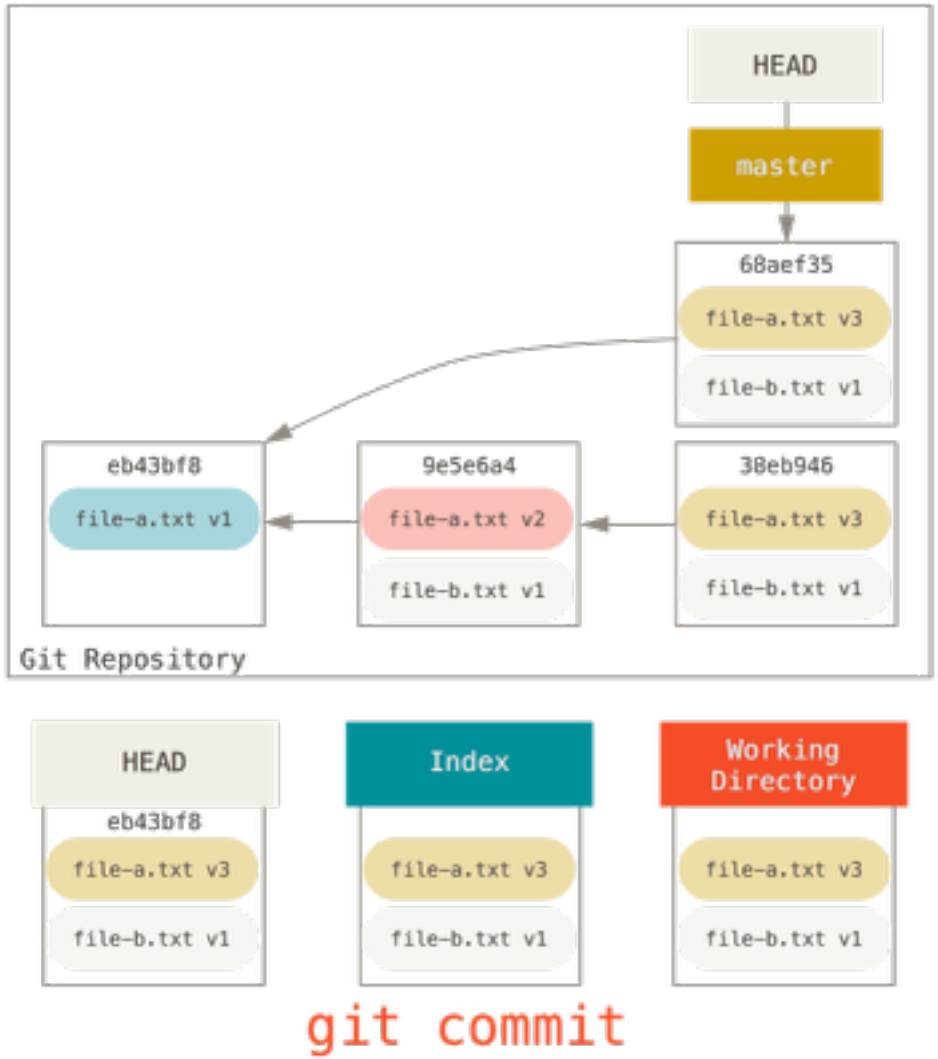
GIT reset squashing



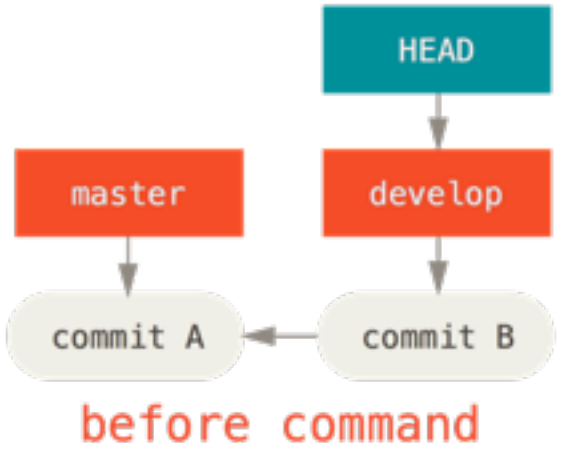


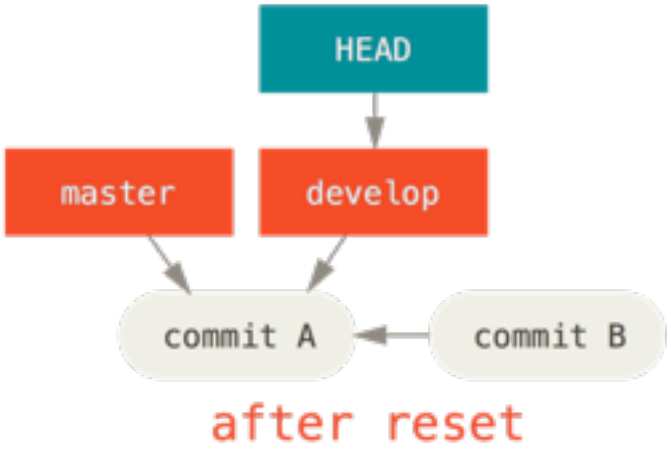

```
git reset --soft HEAD~2
```

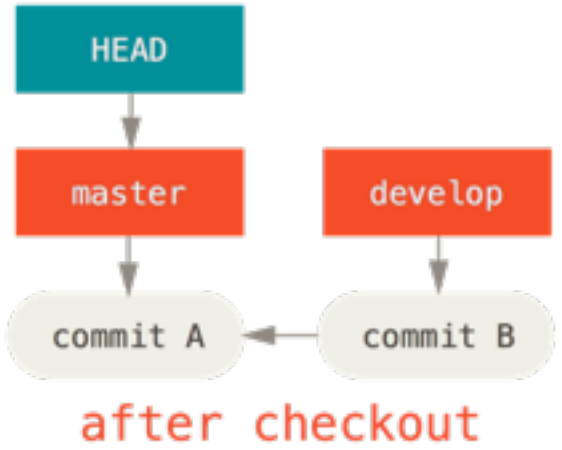
GIT reset squashing



GIT reset vs checkout





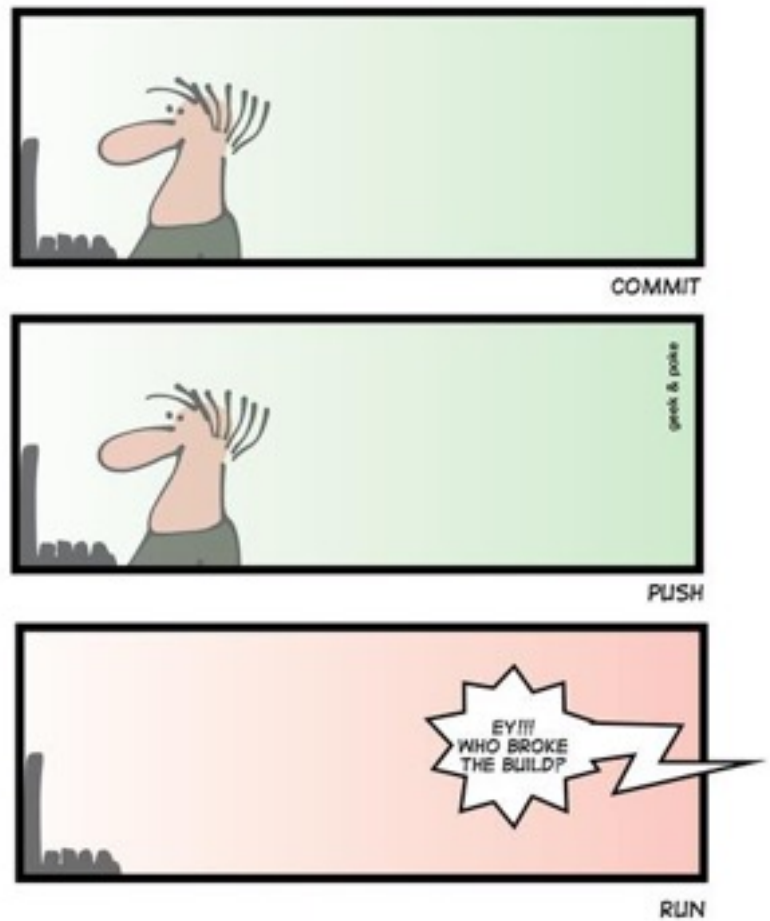


GIT reset summary

	HEAD	Index	Workdir	WD Safe?
Commit Level				
`reset --soft [commit]`	REF	NO	NO	YES
`reset [commit]`	REF	YES	NO	YES
`reset --hard [commit]`	REF	YES	YES	**NO**
`checkout [commit]`	HEAD	YES	YES	YES
File Level				
`reset (commit) [file]`	NO	YES	NO	YES
`checkout (commit) [file]`	NO	YES	YES	**NO**

DEVELOPMENT CYCLE

FRIDAY EVENING EDITION




PUSH REJECTED, REBASE OR MERGE
GIT PUSH --FORCE

Dziękuję za uwagę