

# Git




Иван Домашних

# Базовые понятия

---

# Working directory

---

Имя	Дата изменения	Тип	Размер
 lib	21.02.2016 19:30	Папка с файлами	
 index.html	21.02.2016 19:30	Файл "HTML"	1 КБ
 README	21.02.2016 19:49	Файл	1 КБ

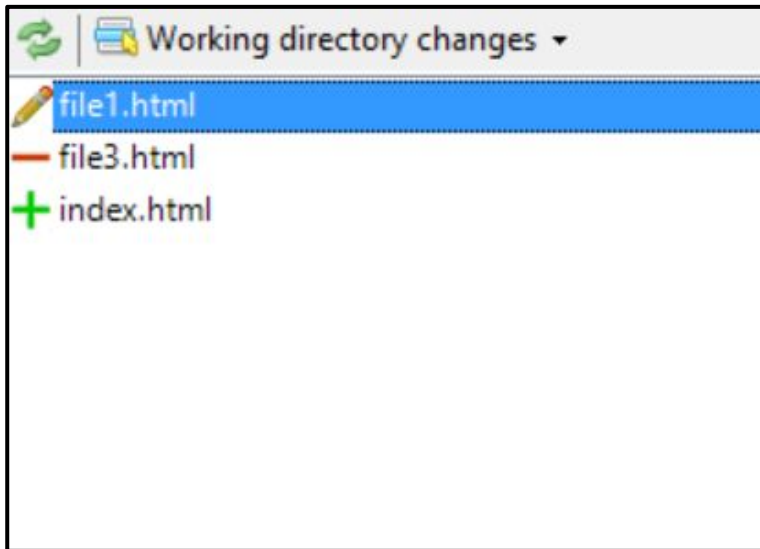
# Modification

---

```
diff --git a/file1.html b/file1.html
index 05c9afd..36f0a56 100644
--- a/file1.html
+++ b/file1.html
@@ -1,2 @@
-<h1>File1</h1>
\ No newline at end of file
+<h1>File1</h1>
+Content
\ No newline at end of file
```

# Commit

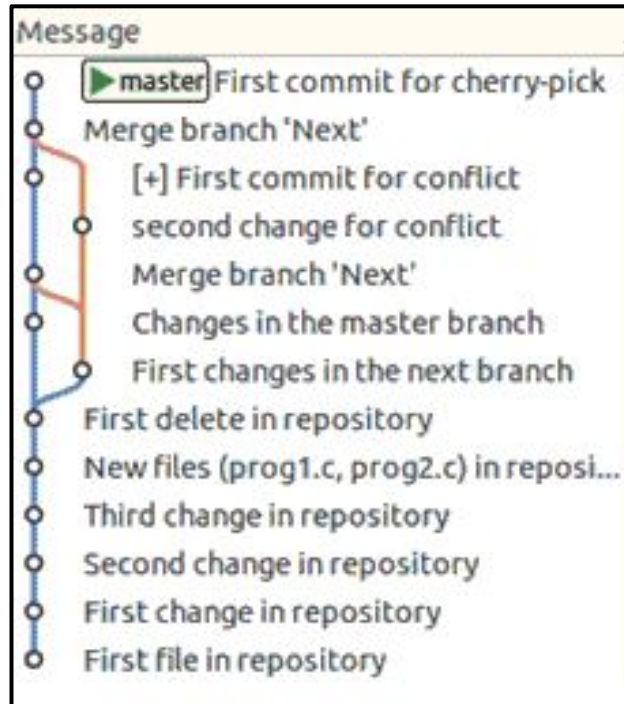
---



Commit hash  
Commit message  
Author  
Date

# History

---



# Repository

---



test/.git – repository + working directory  
test.git – bare repository

# Внесение изменений

---



# add to staging area and commit

---

`git add .`

`git commit`

working directory



```
graph TD; A[working directory] --> B[staging area (index)]; B --> C[repository];
```

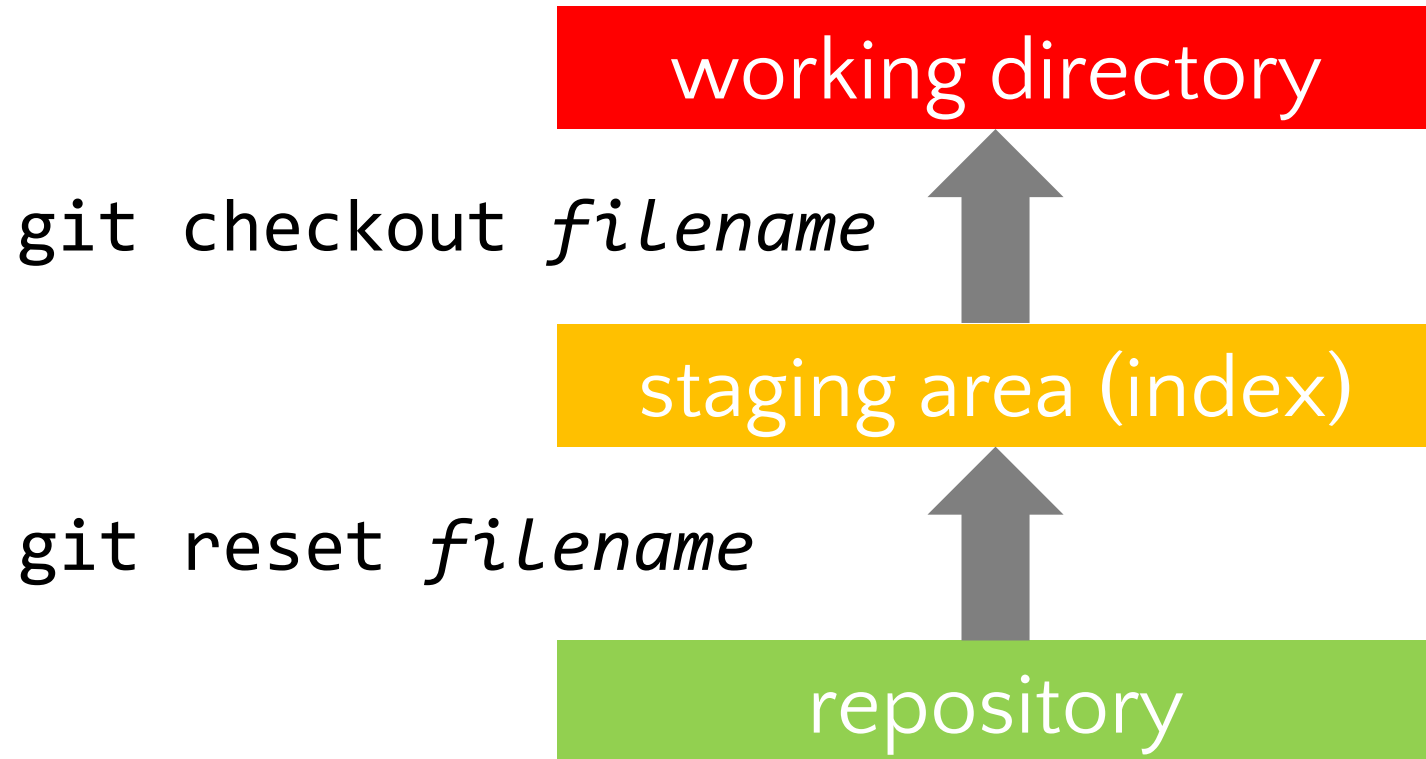
The diagram illustrates the Git workflow. It consists of three rectangular boxes arranged vertically. The top box is red and labeled 'working directory'. A thick grey arrow points down from this box to a yellow box labeled 'staging area (index)'. Another thick grey arrow points down from the yellow box to a green box labeled 'repository'. To the left of the first arrow, the command 'git add .' is written. To the left of the second arrow, the command 'git commit' is written.

staging area (index)

repository

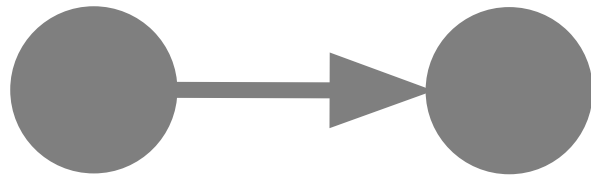
# checkout file or reset file

---

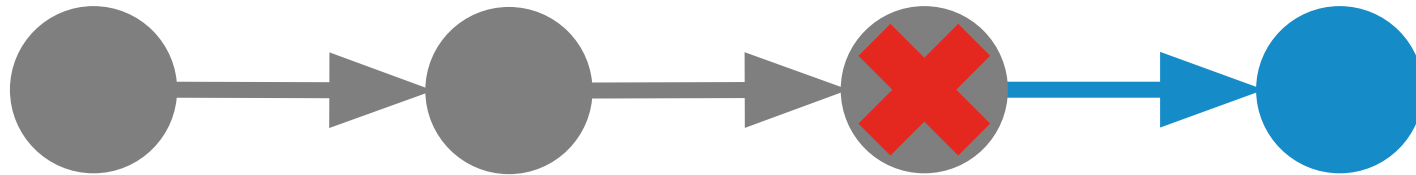


# revert commit

---



||



git revert

# move & remove

---

`git mv` – переименовывает/перемещает файл\*

`git rm` – удаляет файл

\* Git прекрасно умеет отслеживать изменения сам,  
поэтому можно перемещаться файлы любимым способом

# get status of working directory

---

```
# Edit hello.py
git status
# hello.py is listed under "Changes not staged for commit"
git add hello.py
git status
# hello.py is listed under "Changes to be committed"
git commit
git status
# nothing to commit (working directory clean)
```



# log history

---

```
git log -n <limit>  
git log --oneline  
git log --author="<pattern>"
```

```
git log --graph --decorate --oneline  
git log --graph --decorate --oneline --all
```

```
git log --pretty=format:@"%h %ad | %s%d [%an]\  
--graph --date=short
```



# diff

---

`git diff` – разница между `working directory` и `index`

`git difftool` – разница с помощью заданного тула

`git difftool --gui` – разница в GUI-туле

# help

---

git help commit

git commit --help

git commit -h



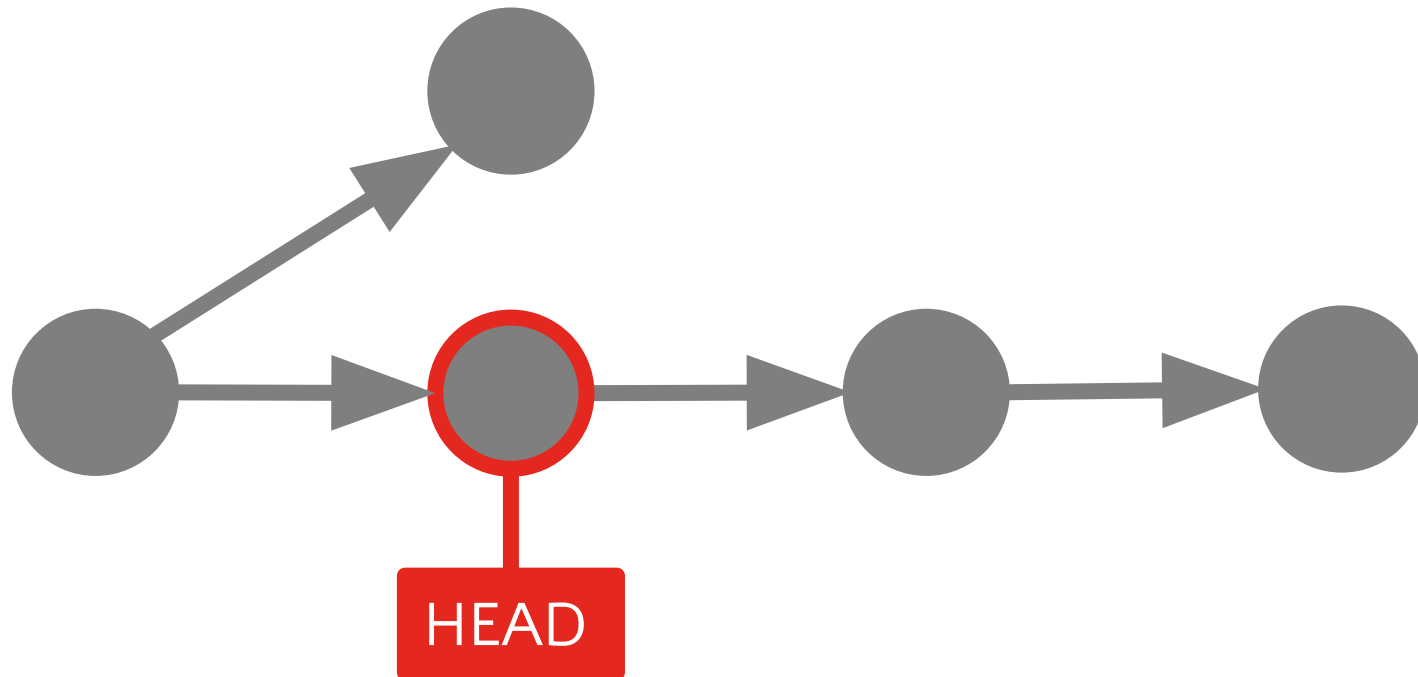


# Метки и ветки

---

# HEAD

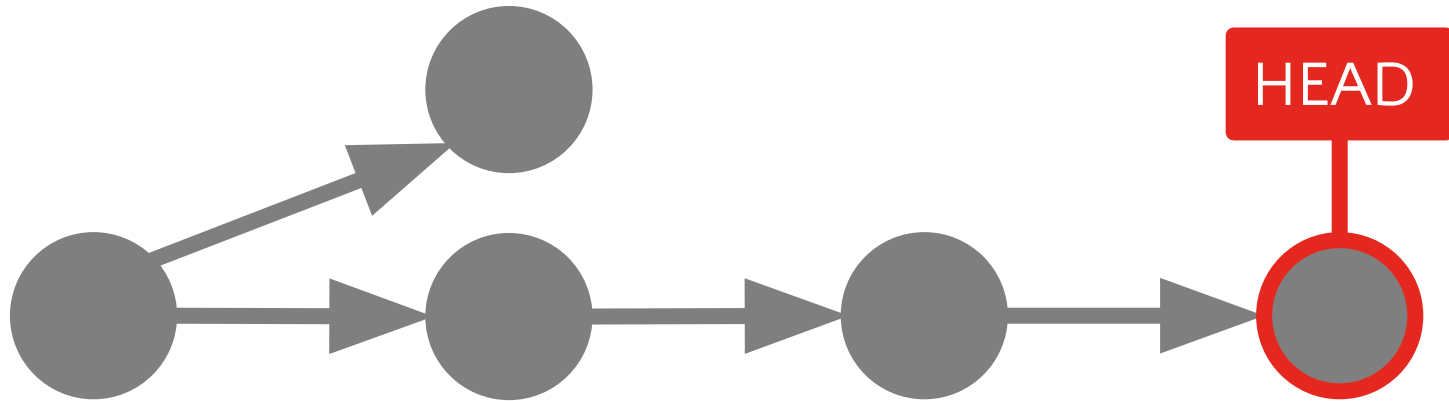
---



# checkout to revision

---

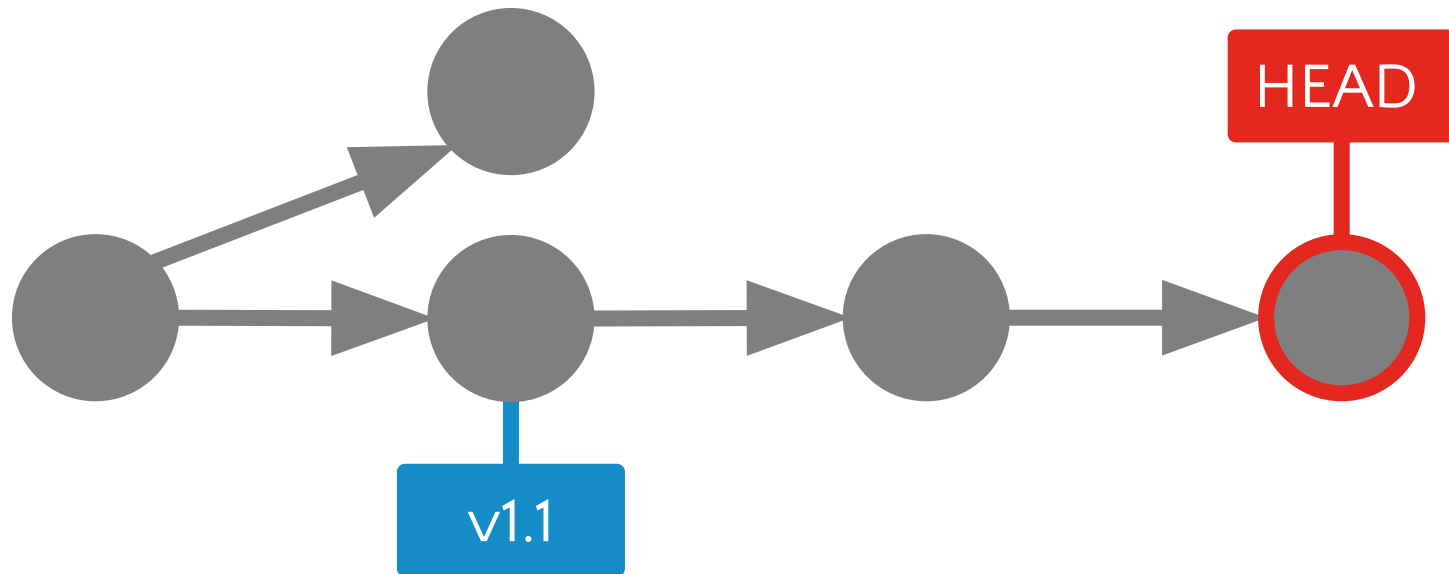
```
git checkout <revision>
```



# tag

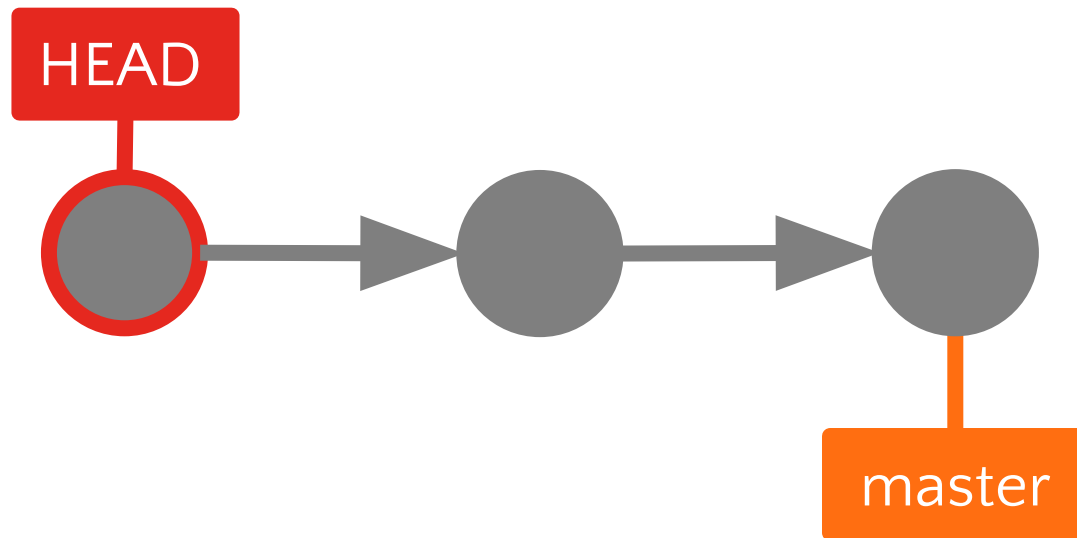
---

```
git tag "v1.1"
```



# branch

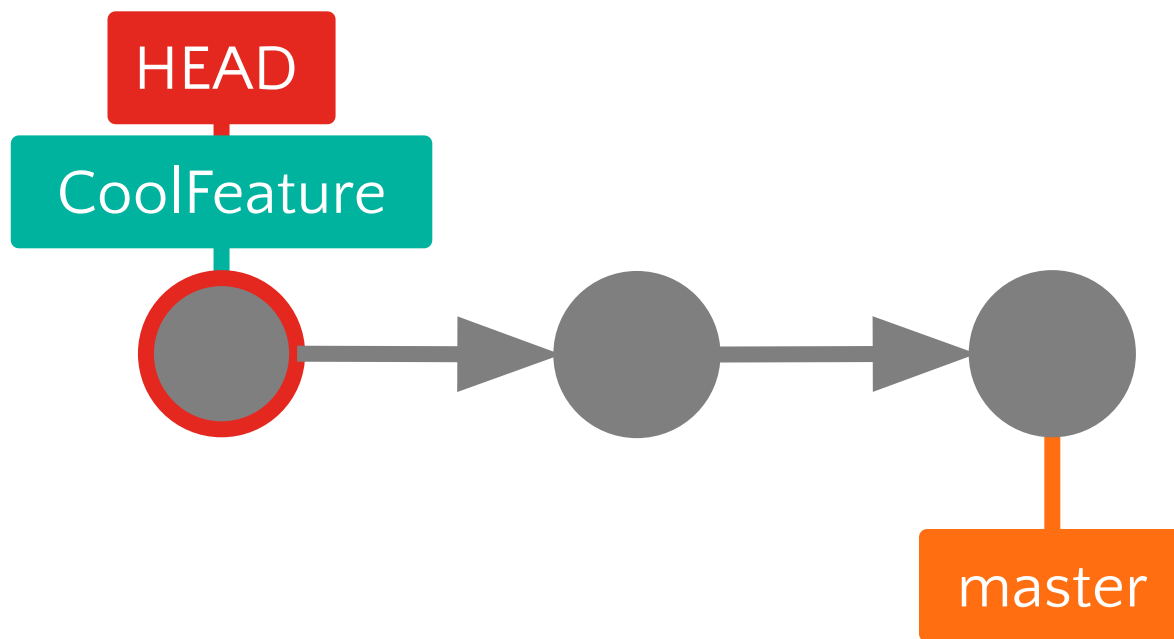
---



master – ветка по умолчанию

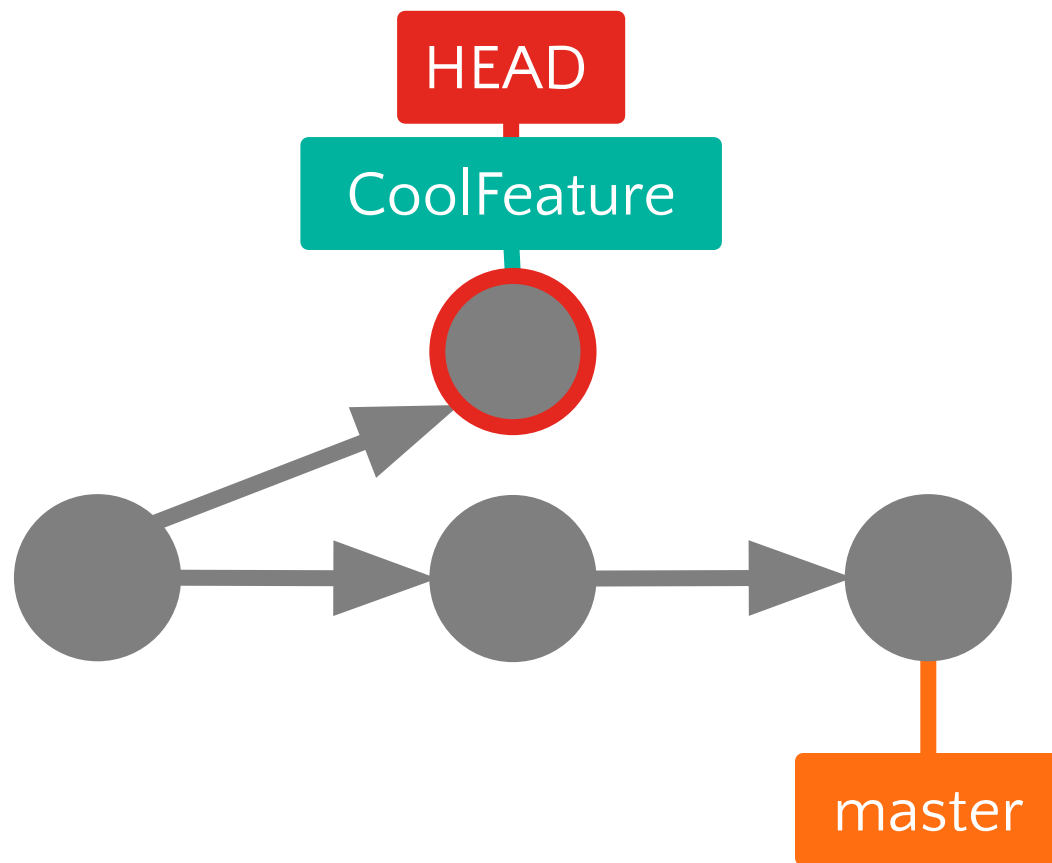
# Развитие фичи

---



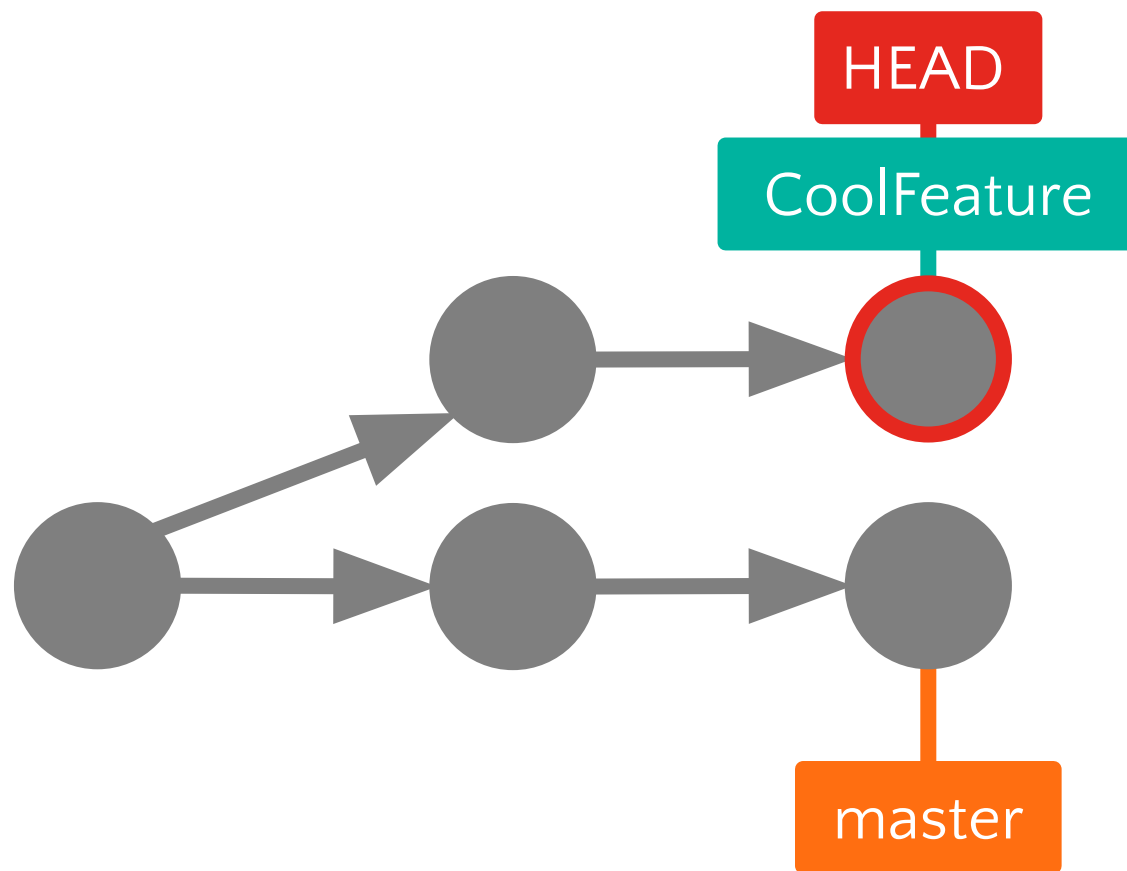
# Развитие фичи

---



# Развитие фиичи

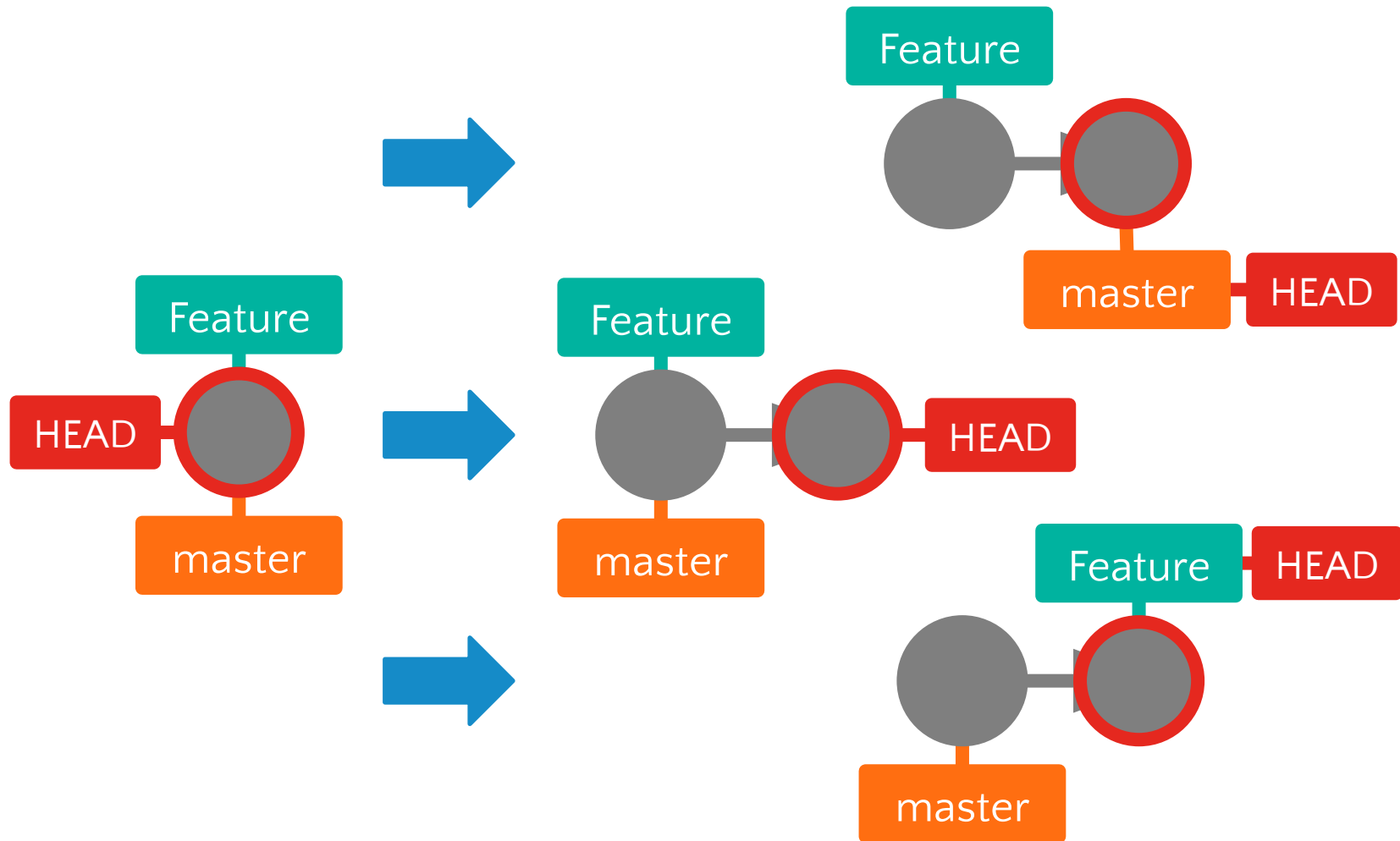
---





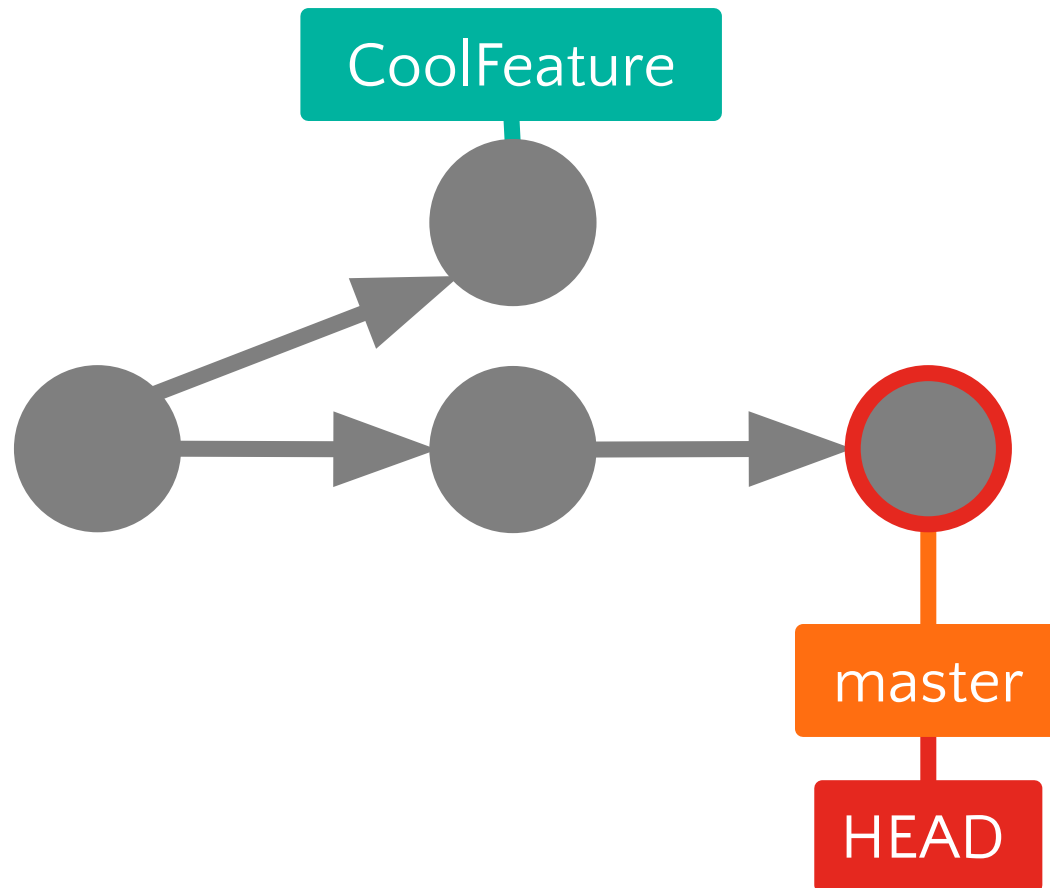
# Что будет при коммите?

---



# checkout branch

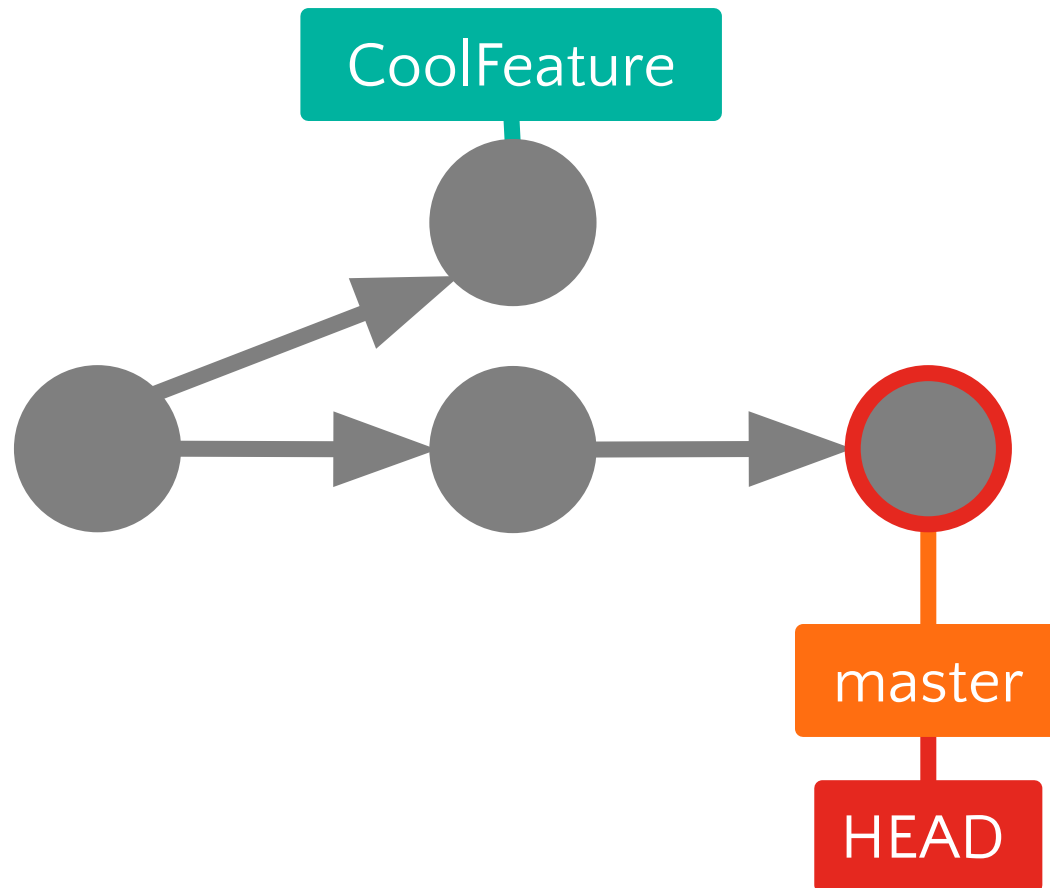
---



# checkout branch

---

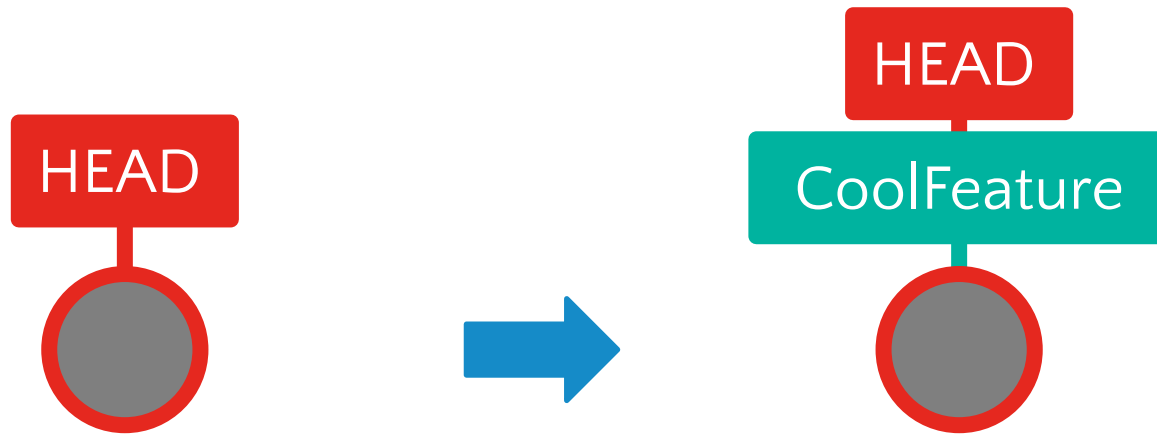
```
git checkout CoolFeature
```



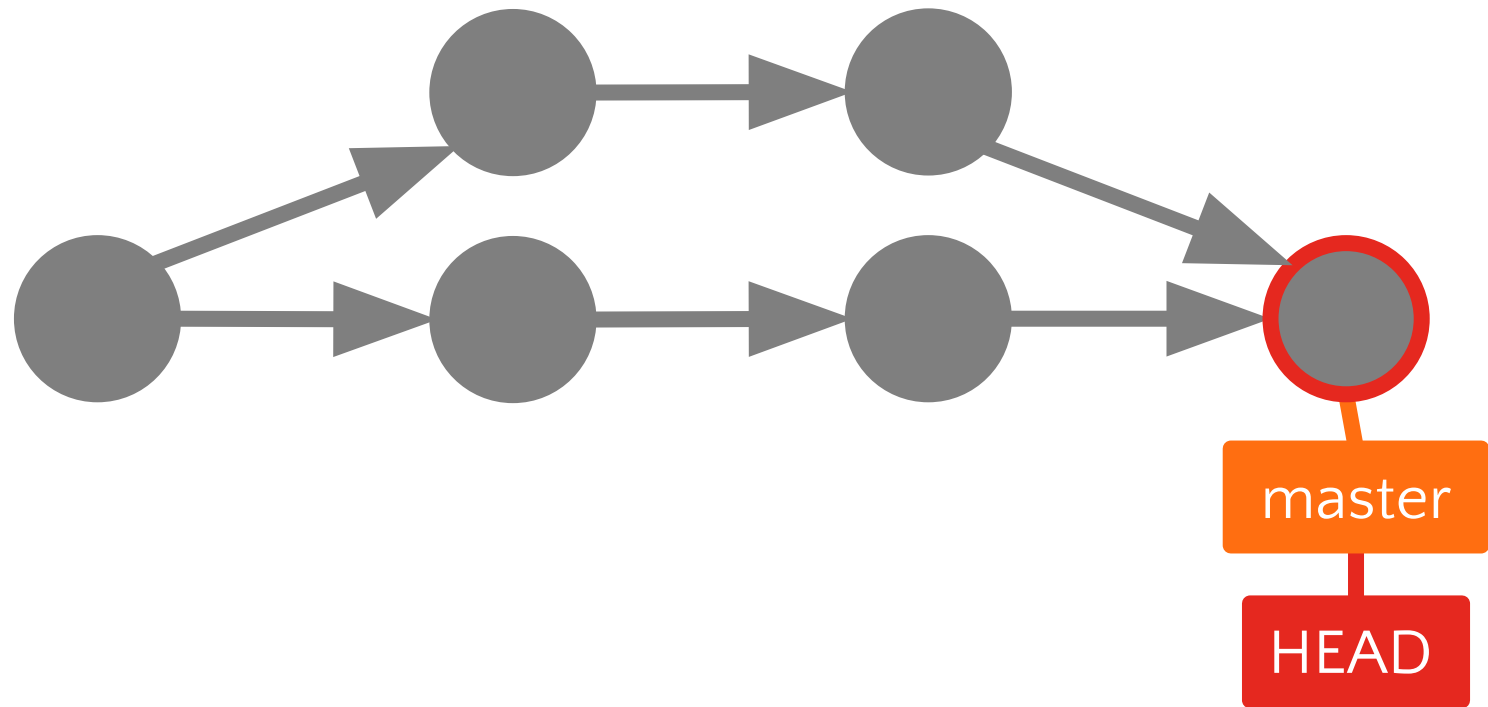
# create and checkout branch

---

```
git checkout -b CoolFeature
```

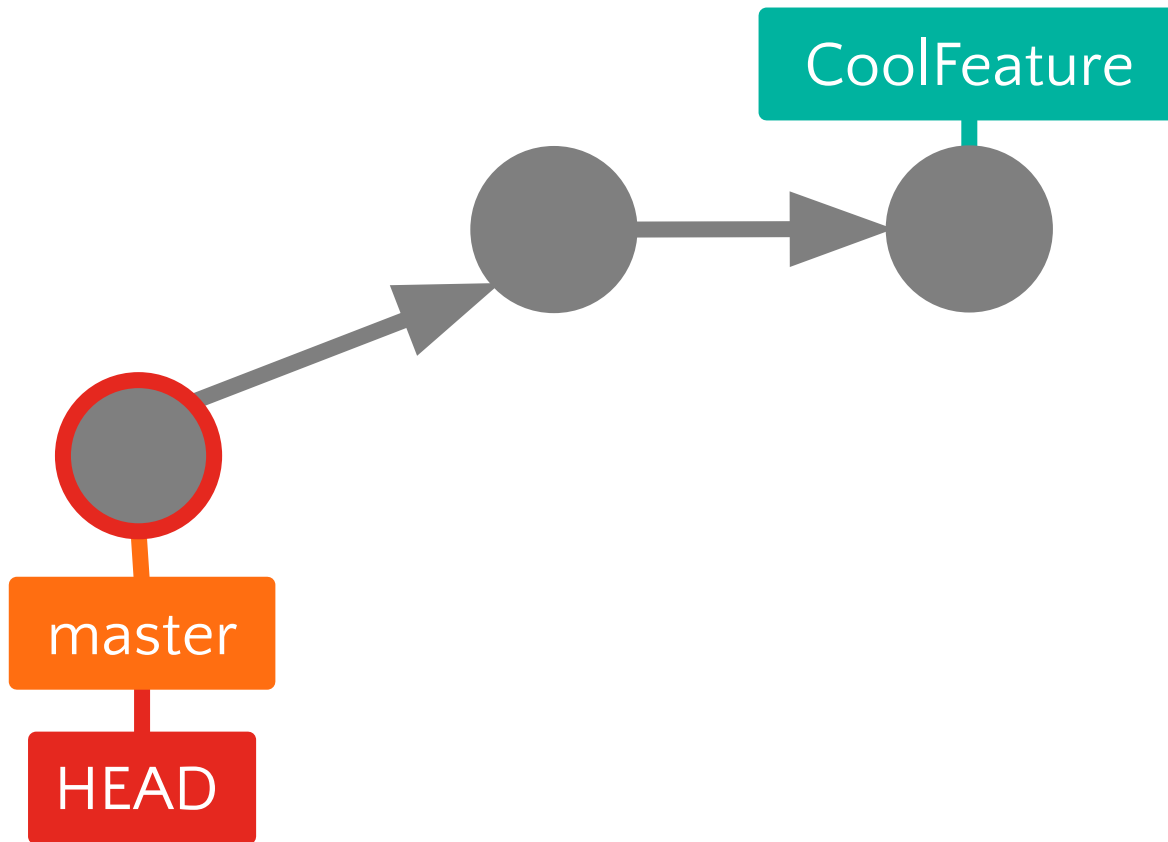


# merge



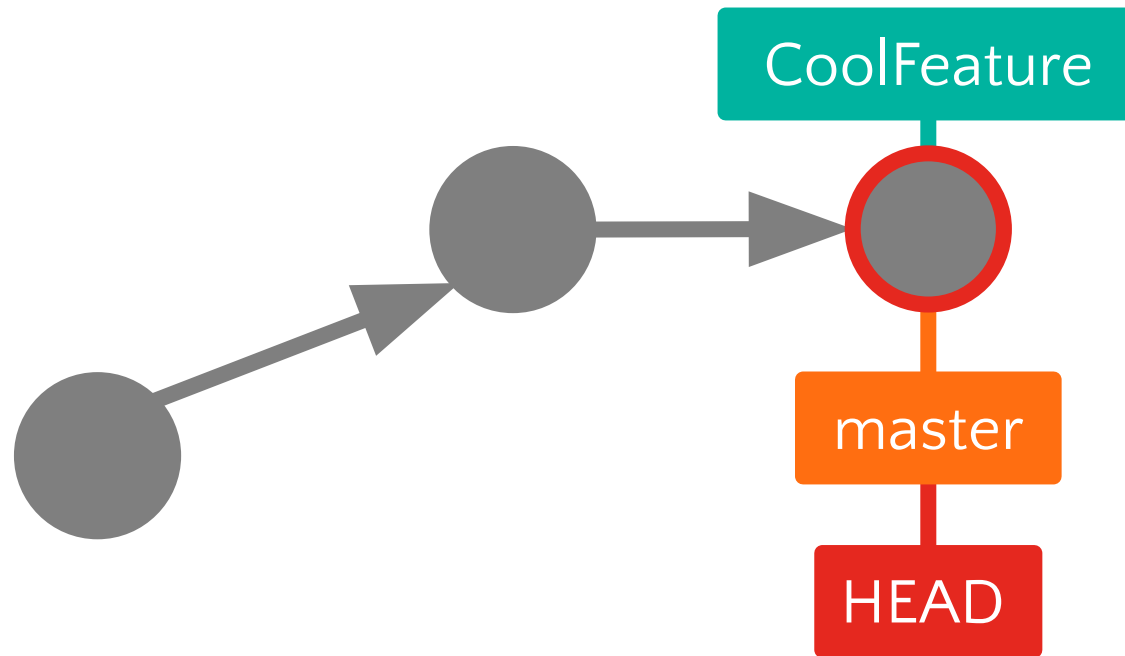
# Fast Forward Merge

---



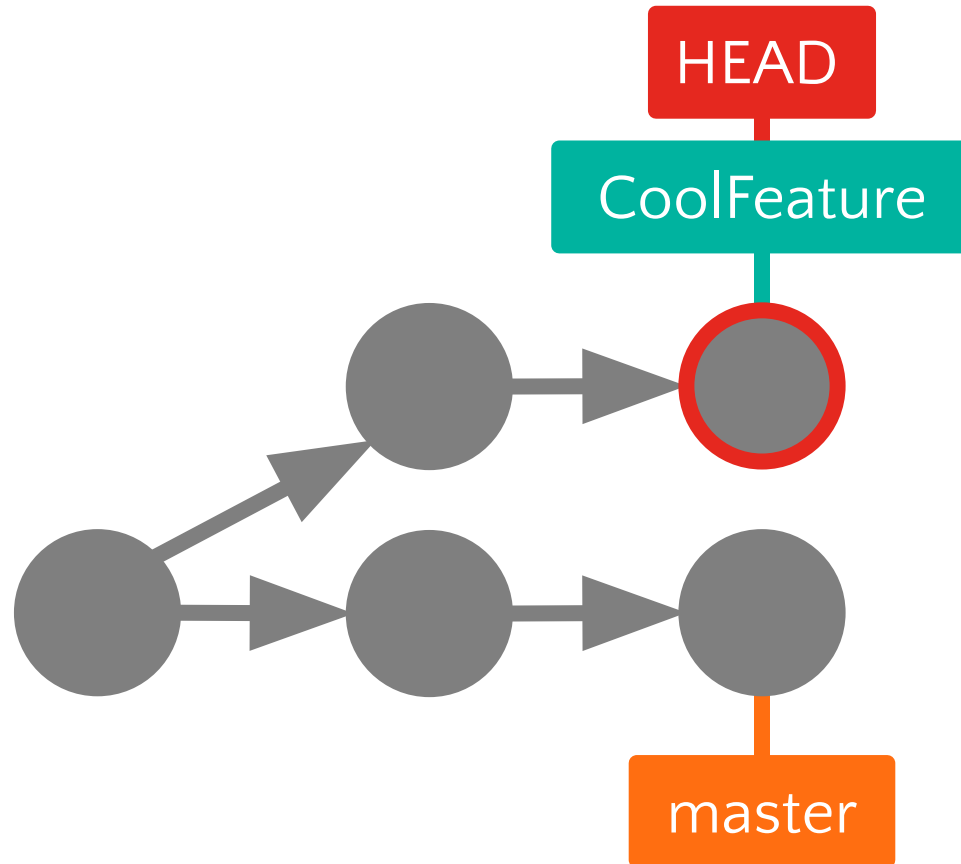
# Fast Forward Merge

---



# rebase

---

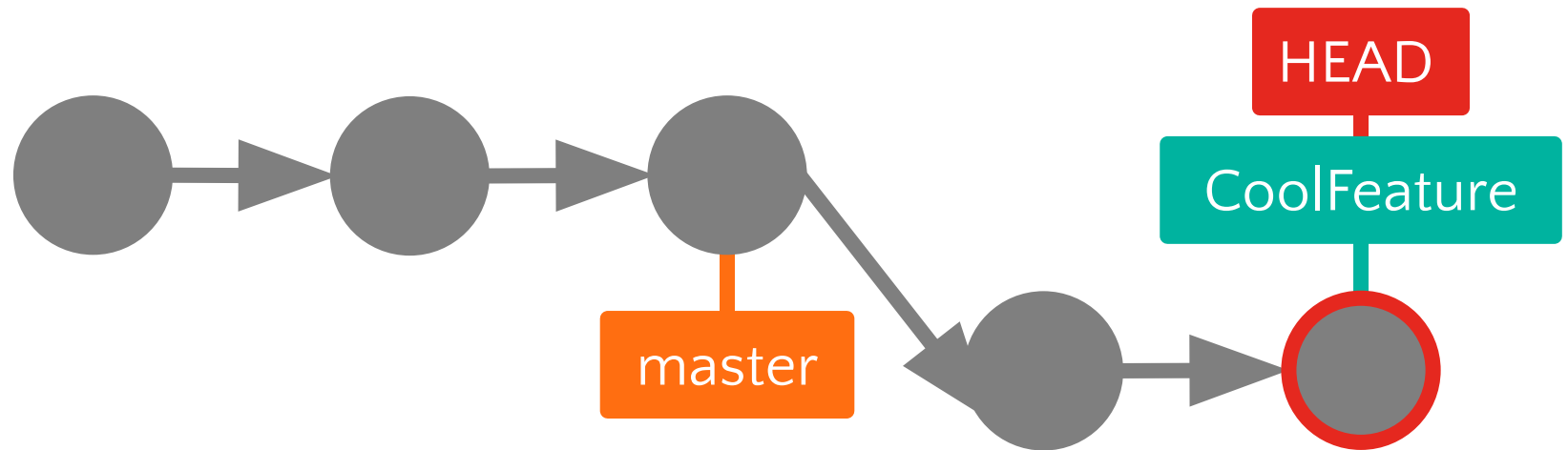




# rebase

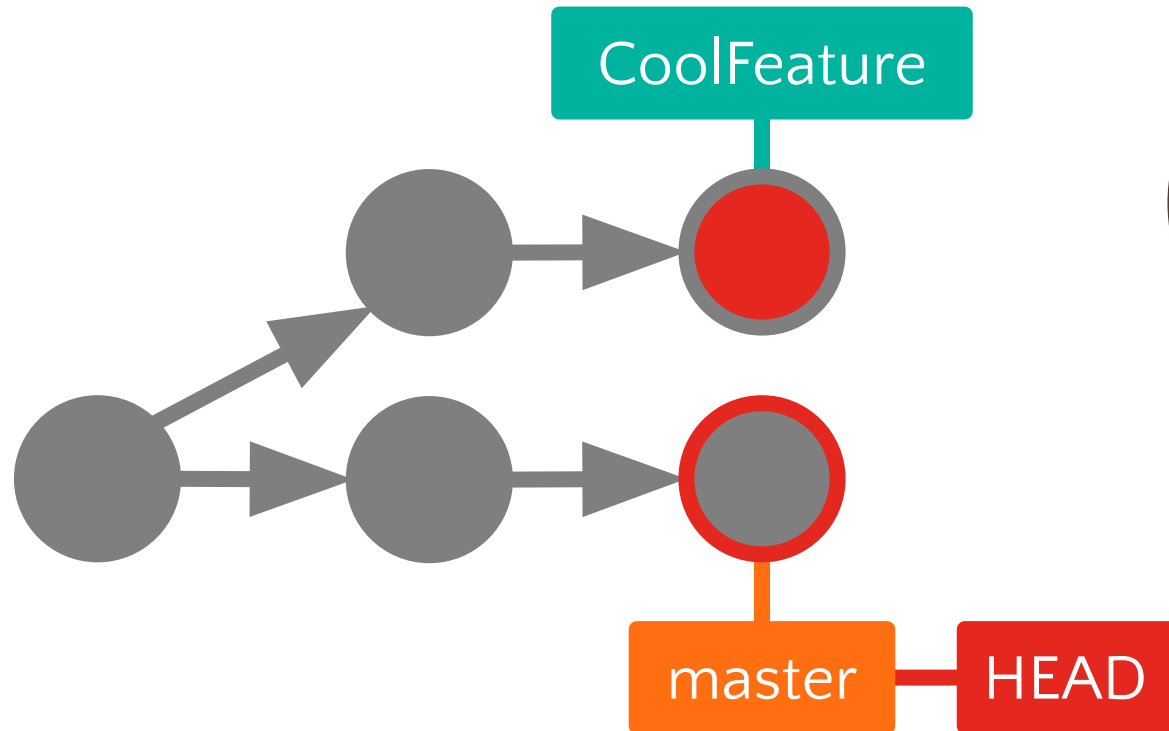
---

*Achtung!* Команда меняет историю



# cherry-pick

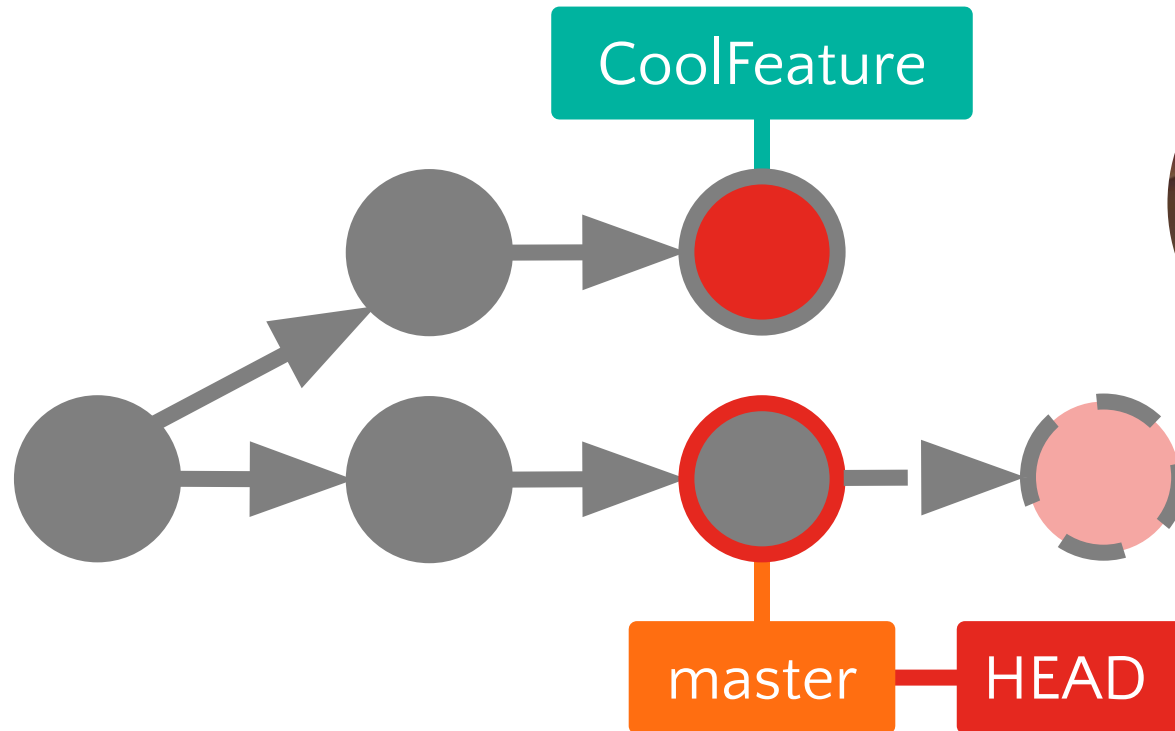
---



# cherry-pick

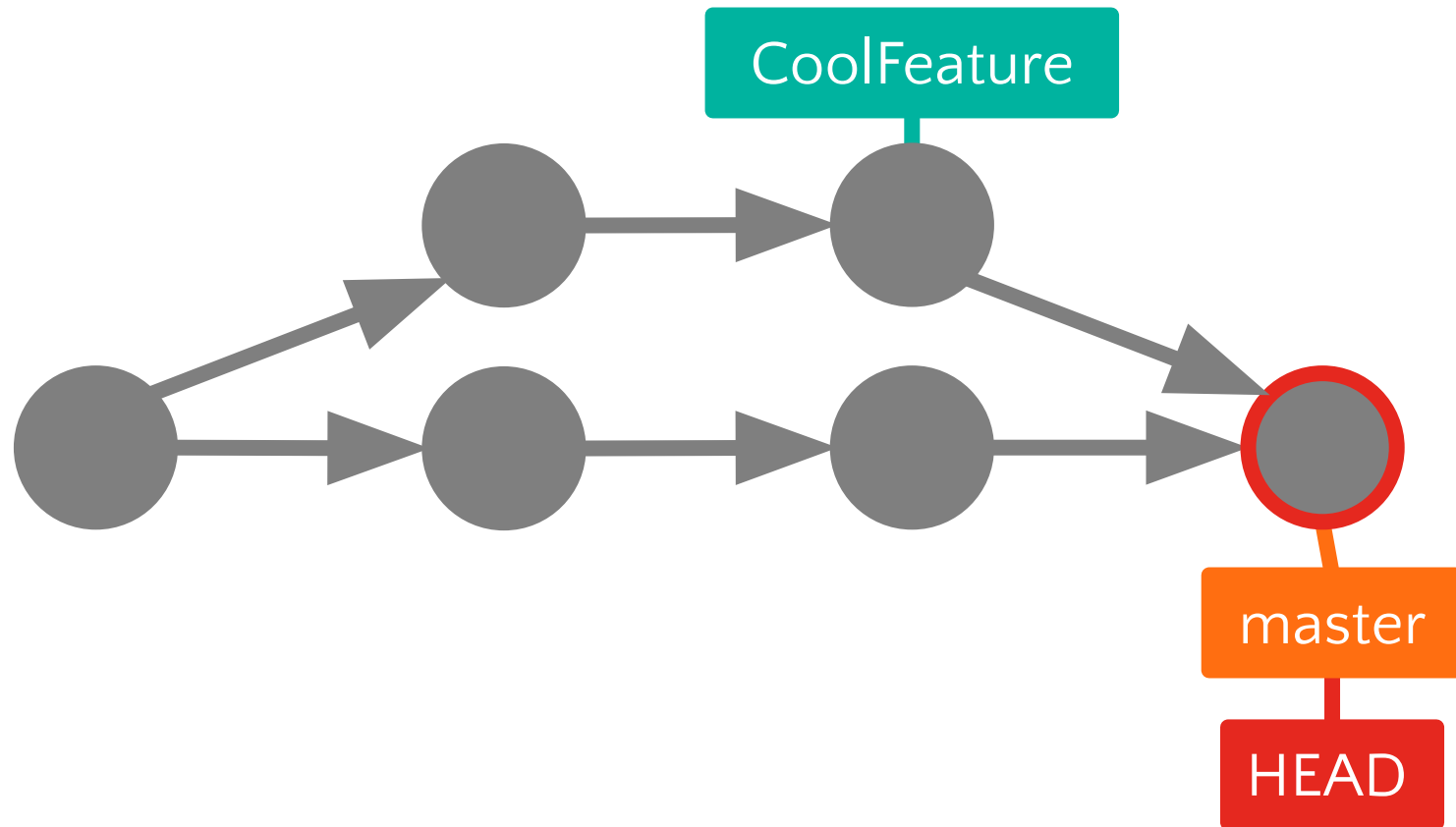
---

```
git cherry-pick <commit-hash>
```



# branch delete

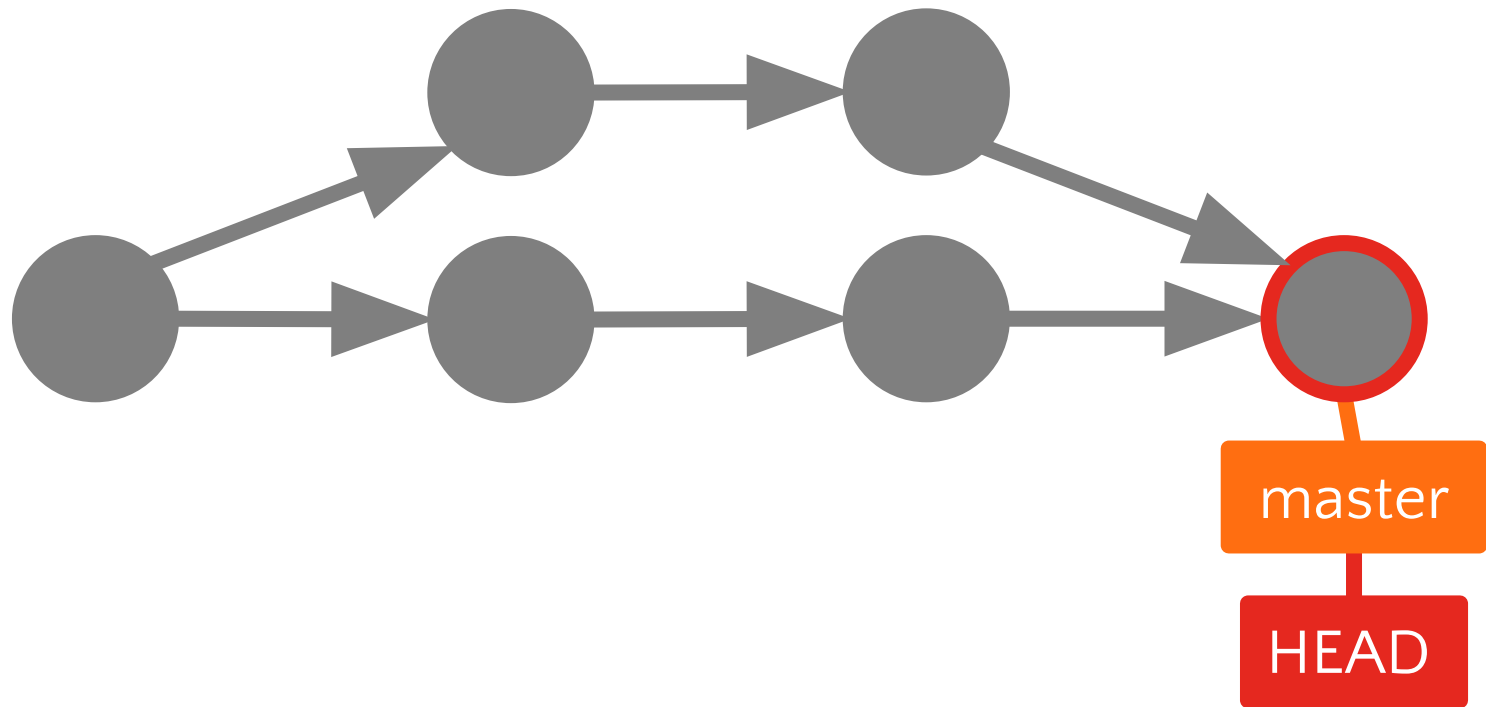
---



# branch delete

---

```
git branch -d "CoolFeature"
```



# Конфликт

---

```
<<<<<< HEAD:index.html
<div id="footer">contact :
email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```

# Reflog, gc

---

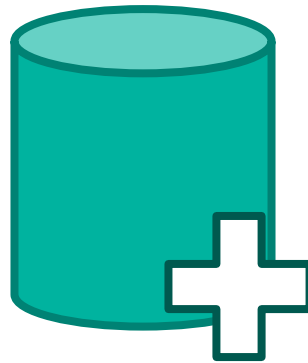
# Операции над репозиториями

---



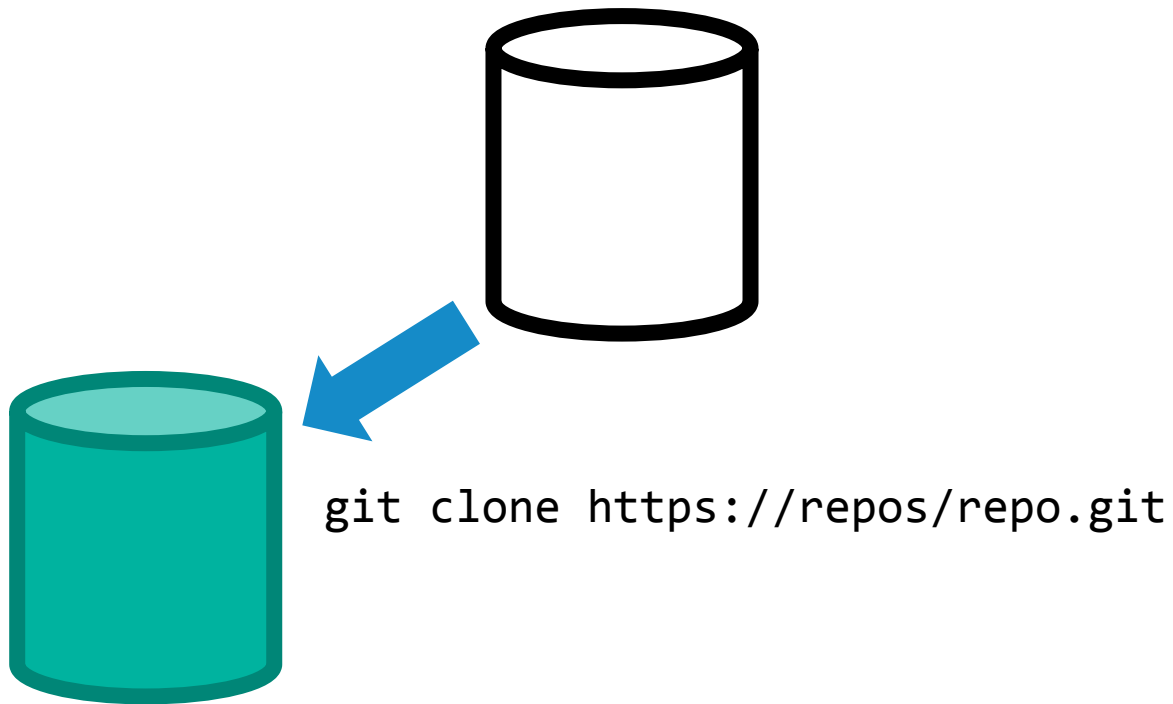
# init

---



# clone

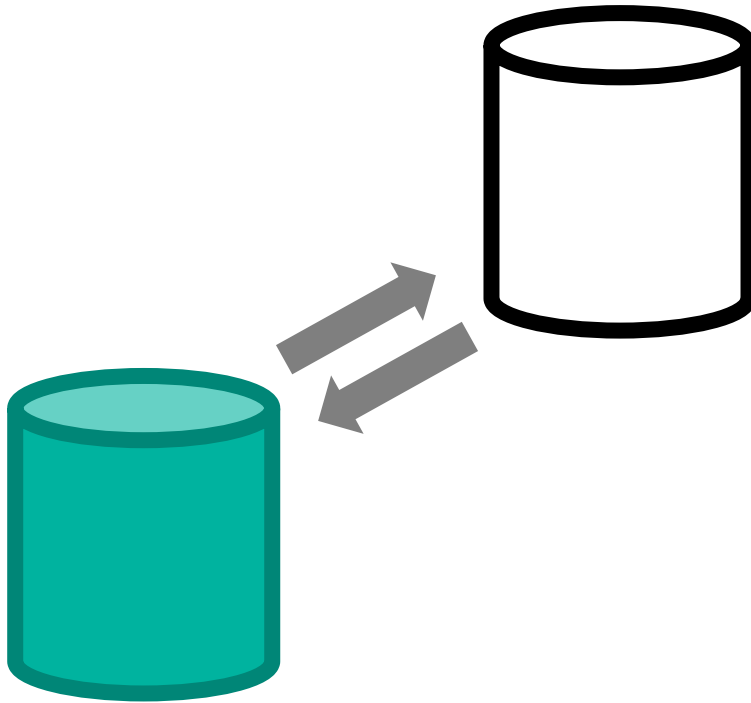
---



# Remotes

---

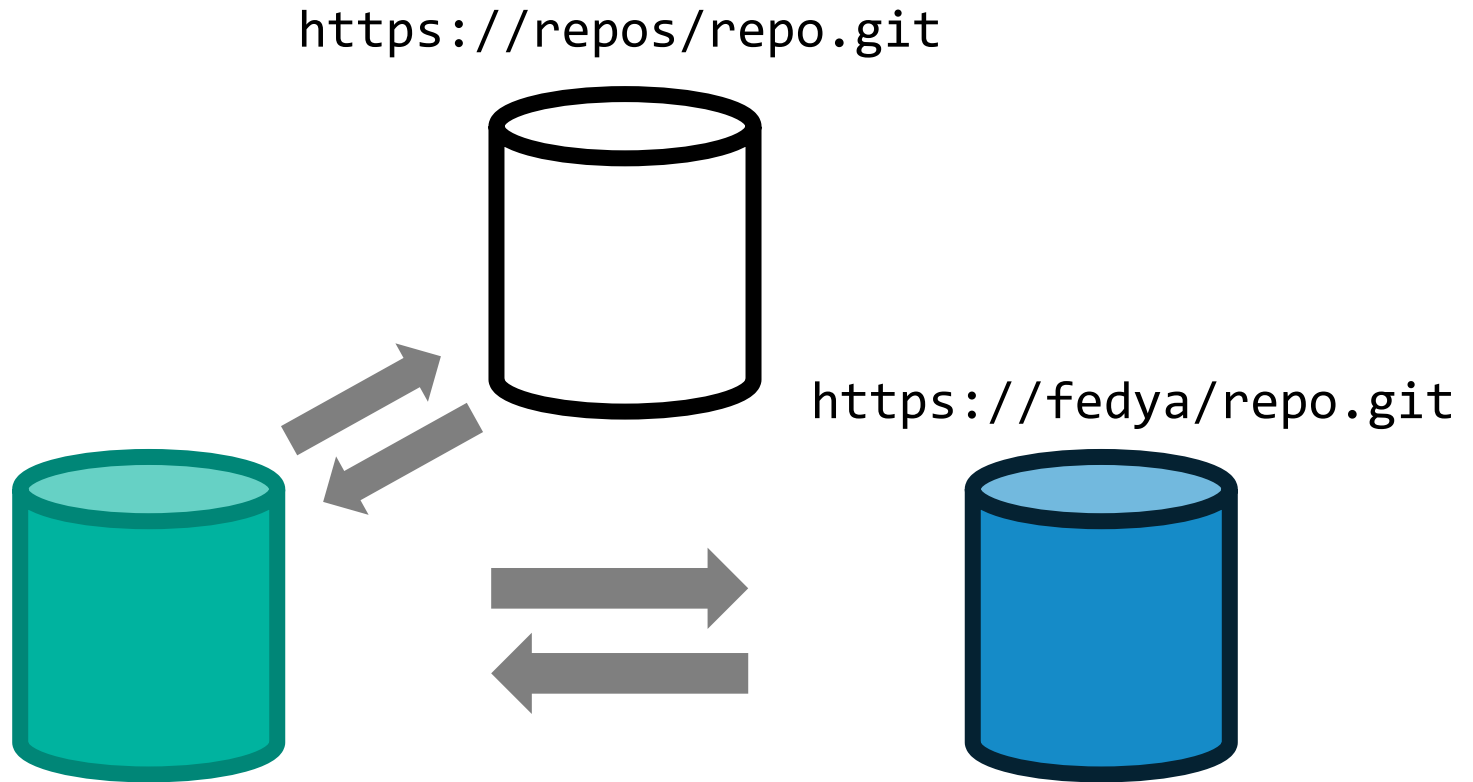
`https://repos/repo.git`



`origin = https://repos/repo.git`

# Remotes

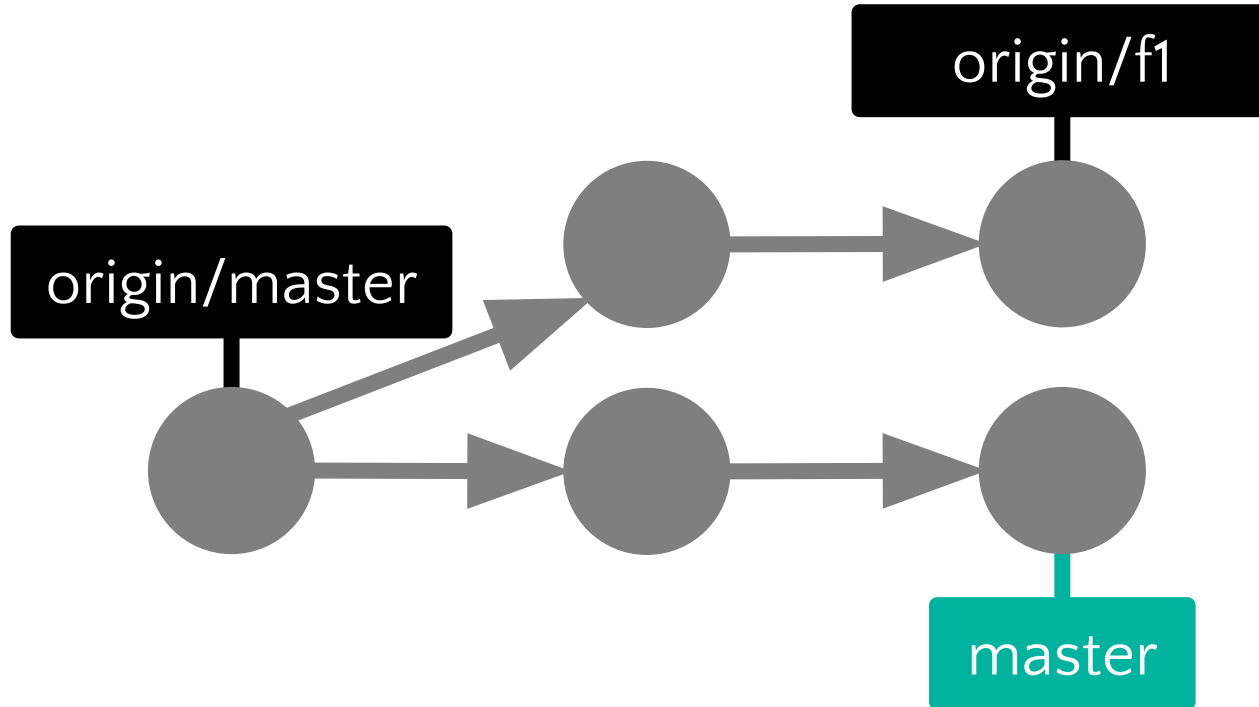
---



`origin = https://repos/repo.git`  
`fedya = https://fedya/repo.git`

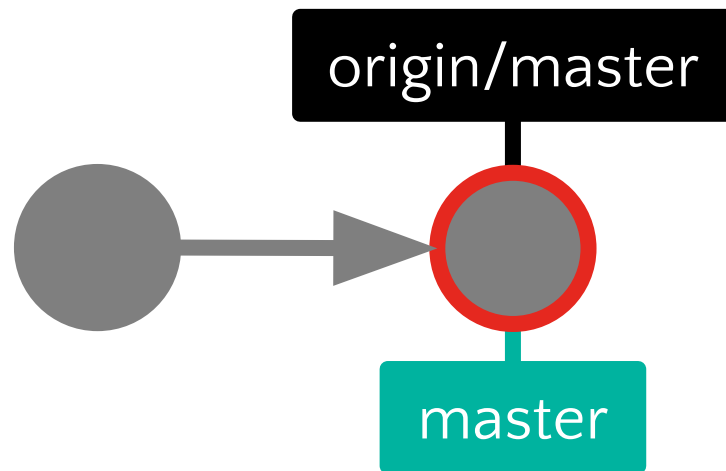
# Remote branches

---



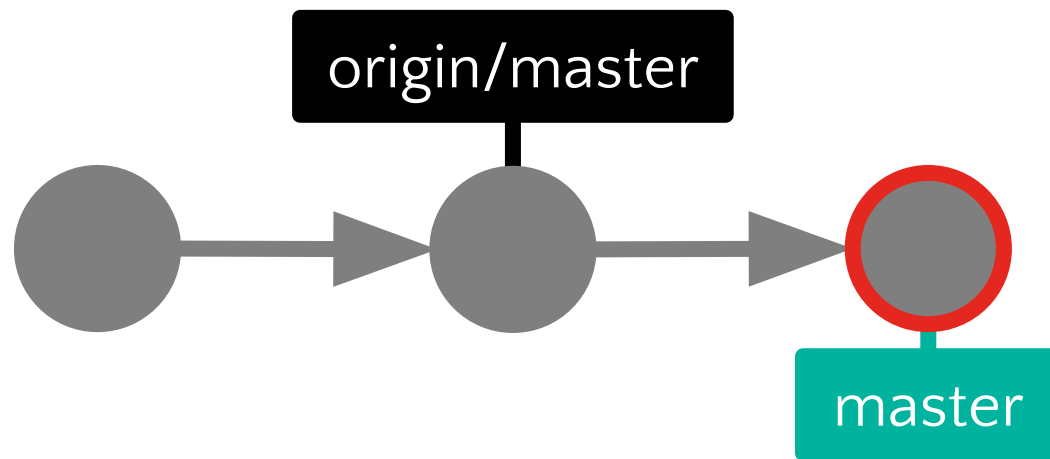
# fetch

---



# fetch

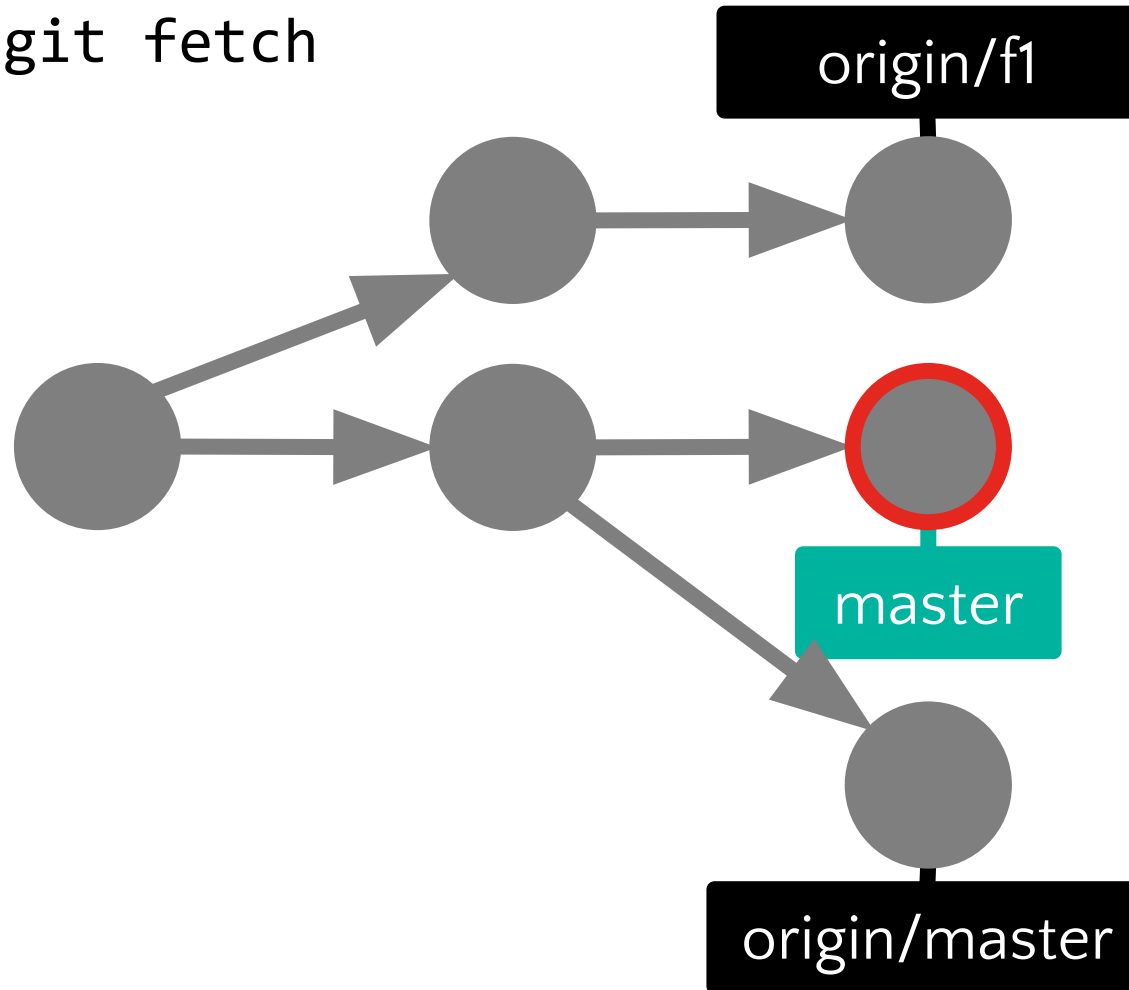
---



# fetch

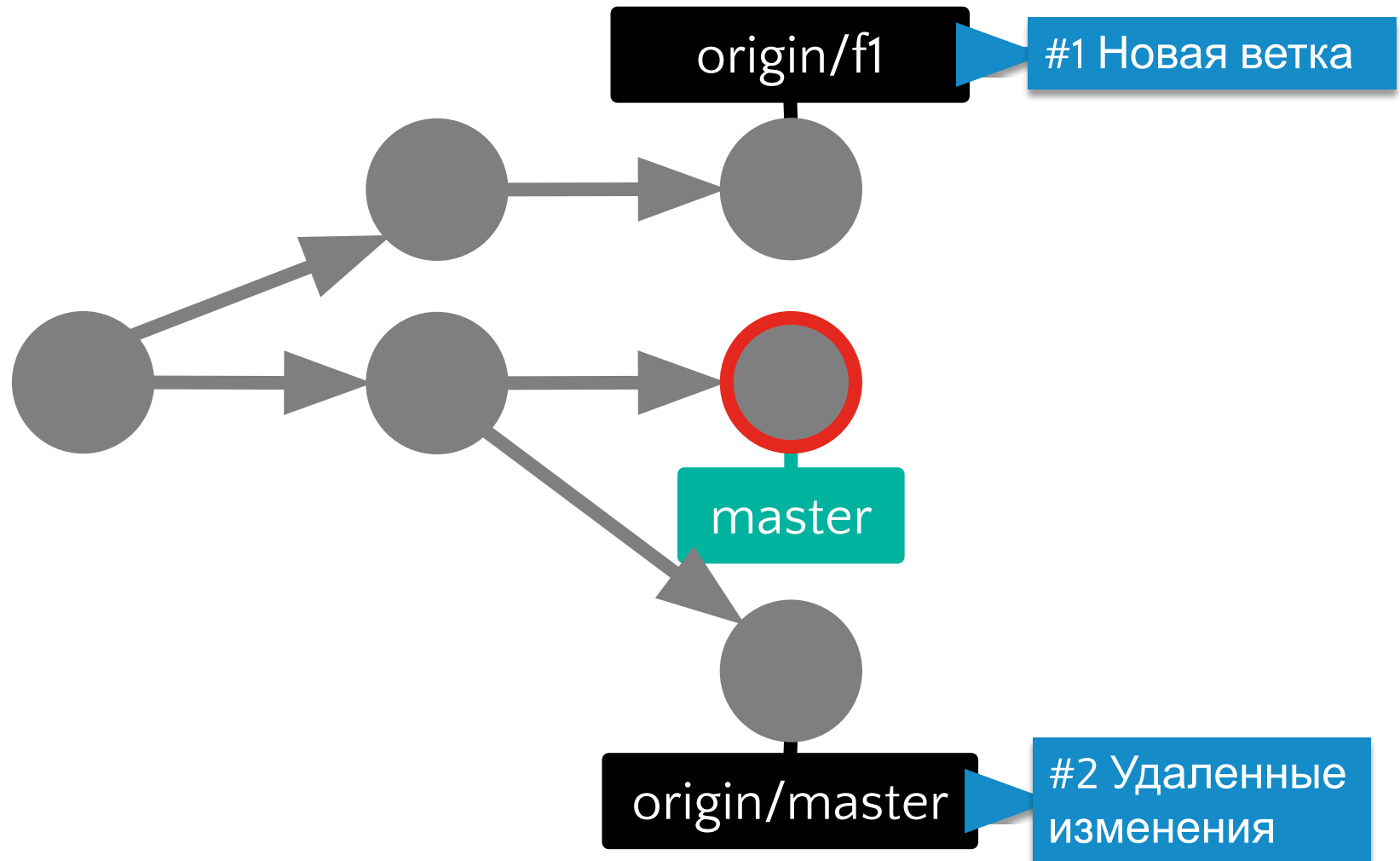
---

git fetch



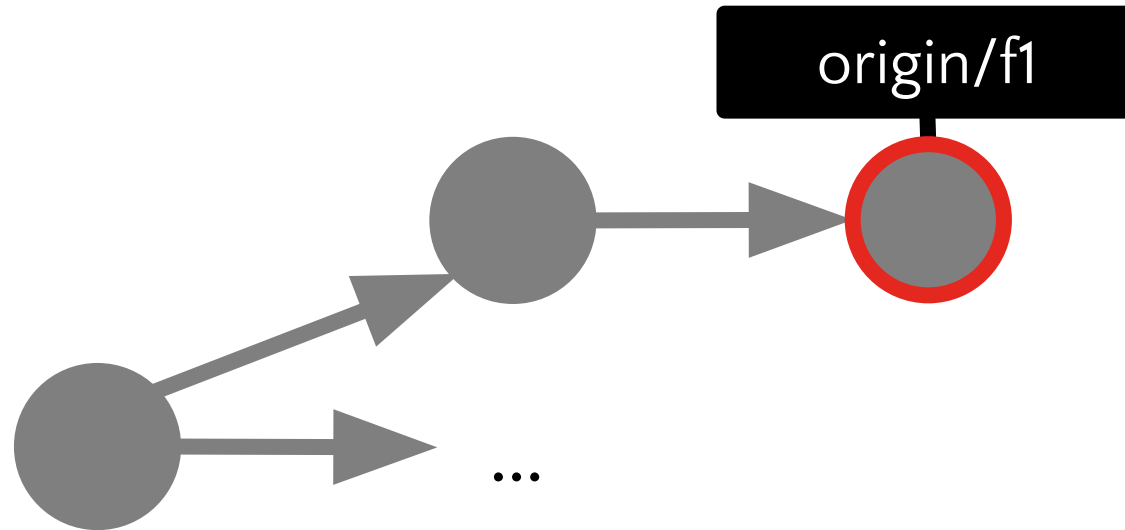


# Что делать?



# #1 Новая ветка

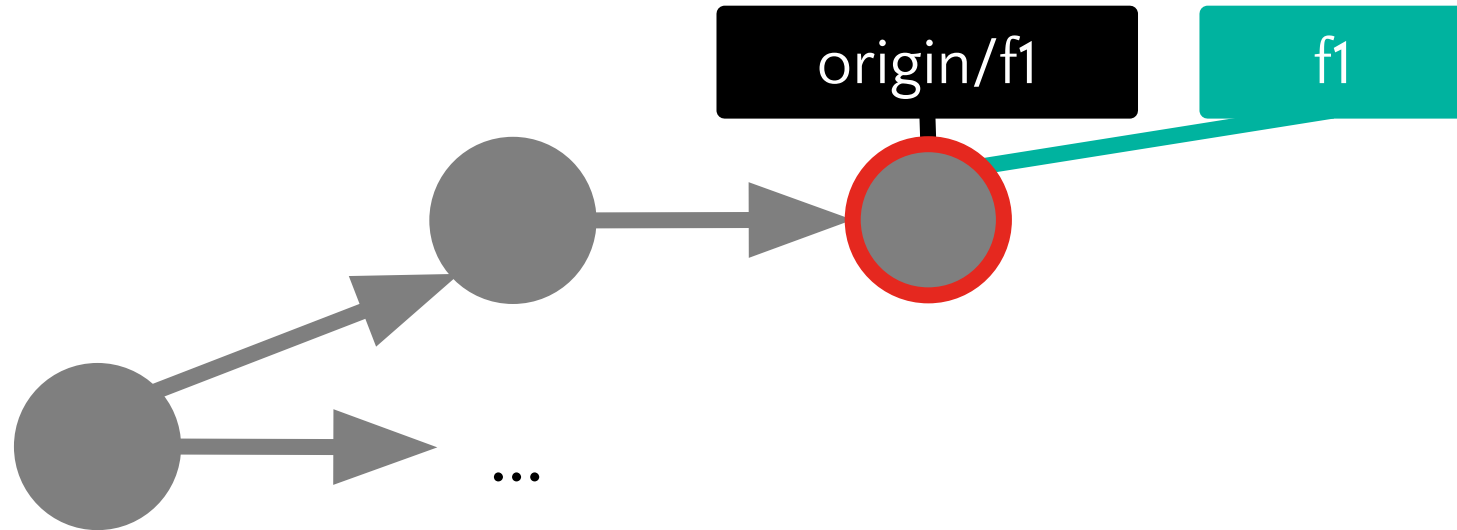
---



`git checkout -b f1` ?

# #1 Новая ветка

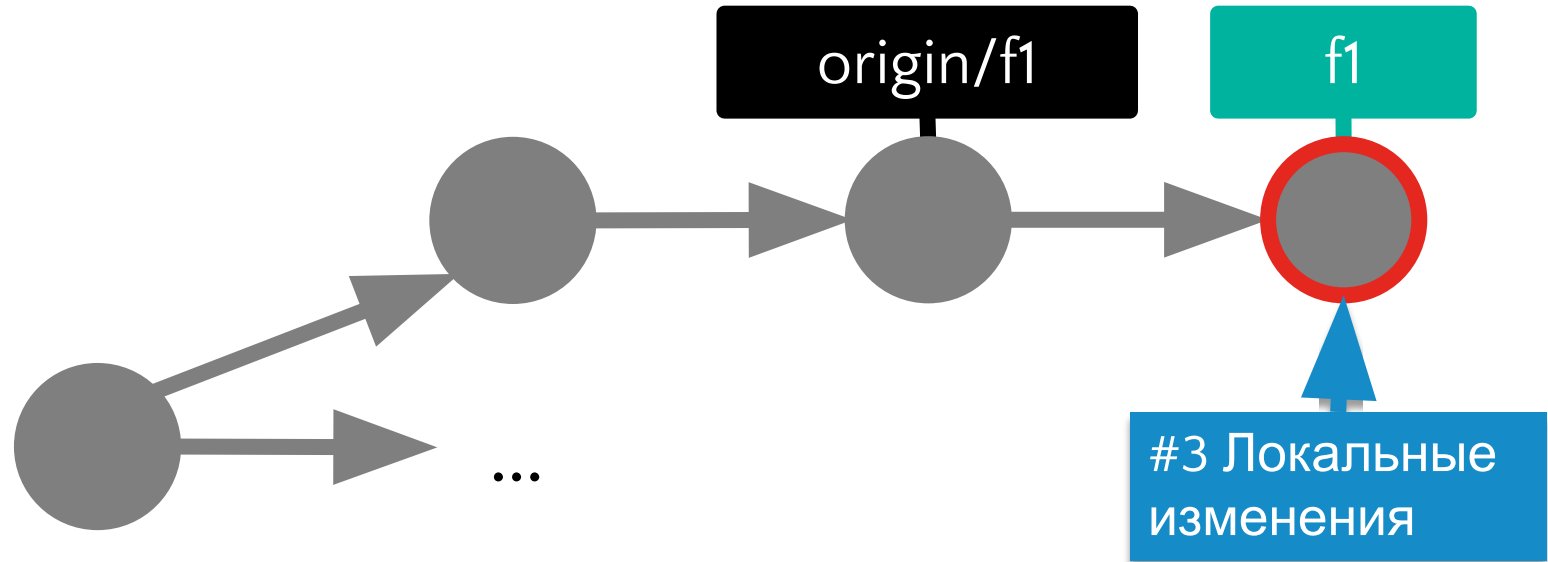
---



`git checkout -b f1` ?

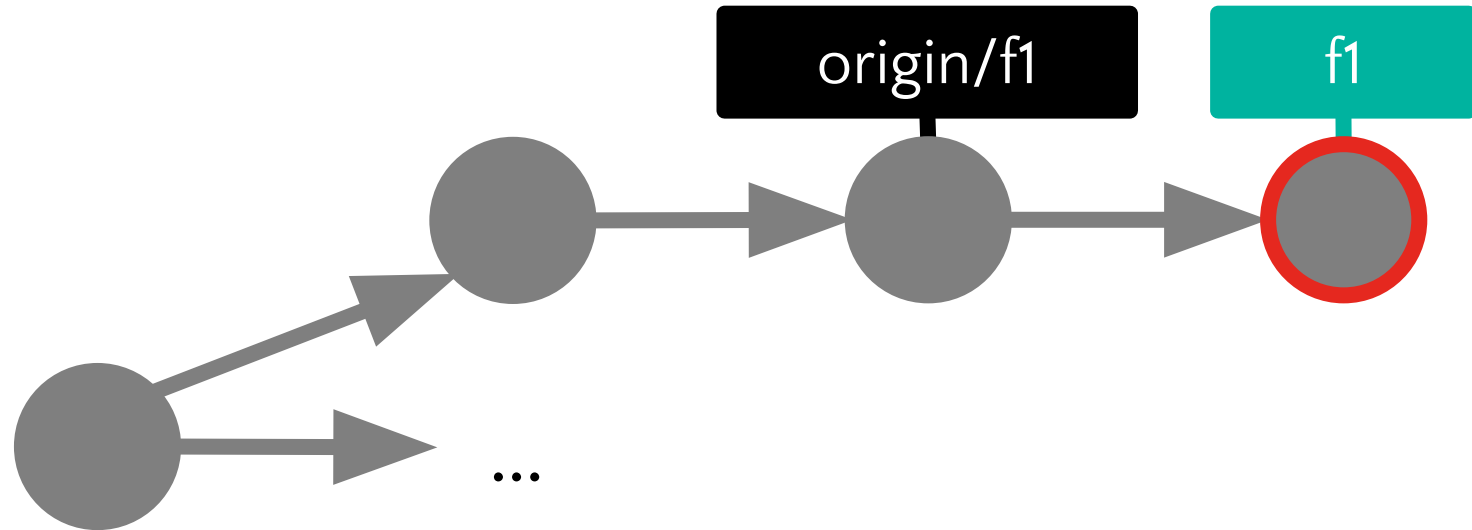
# Начинаем править

---



# #3 Локальные изменения

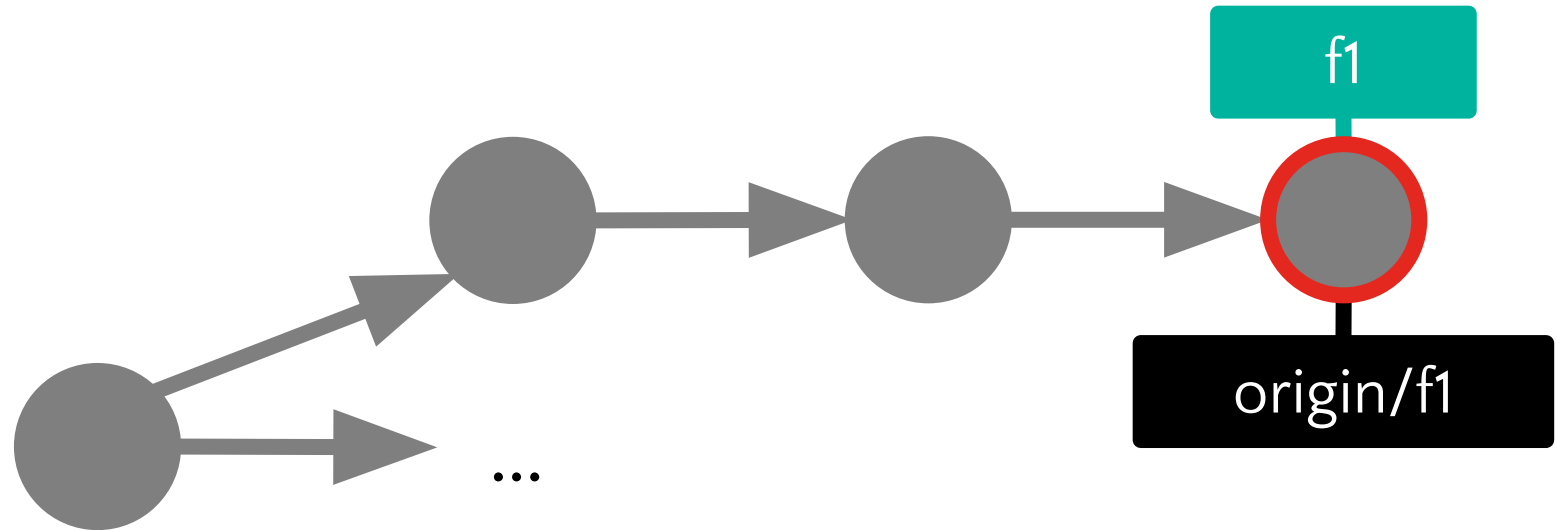
---



git ?

# #3 Локальные изменения

---



git ?

# Таблица веток

---

```
$ git branch -vv
```

```
f1      7e424c3 [origin/f1]
```

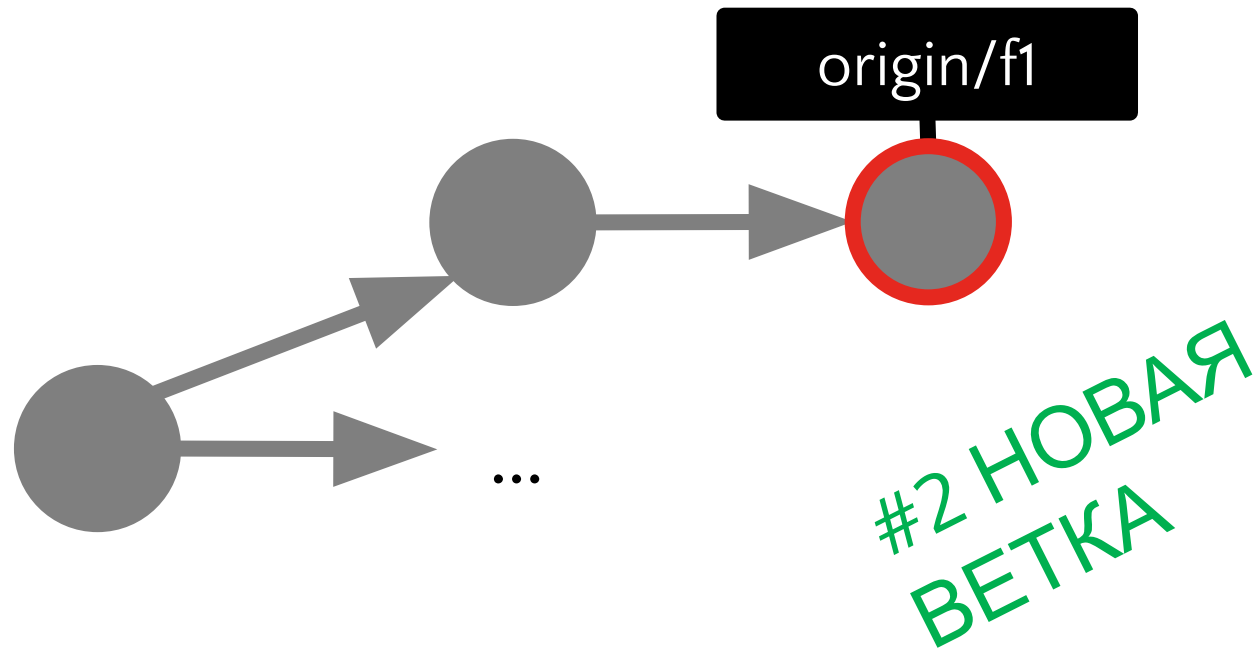
```
master 1ae2a45 [origin/master]
```

```
f2      5ea463a
```

upstream branch or tracking branch

# checkout upstream

---



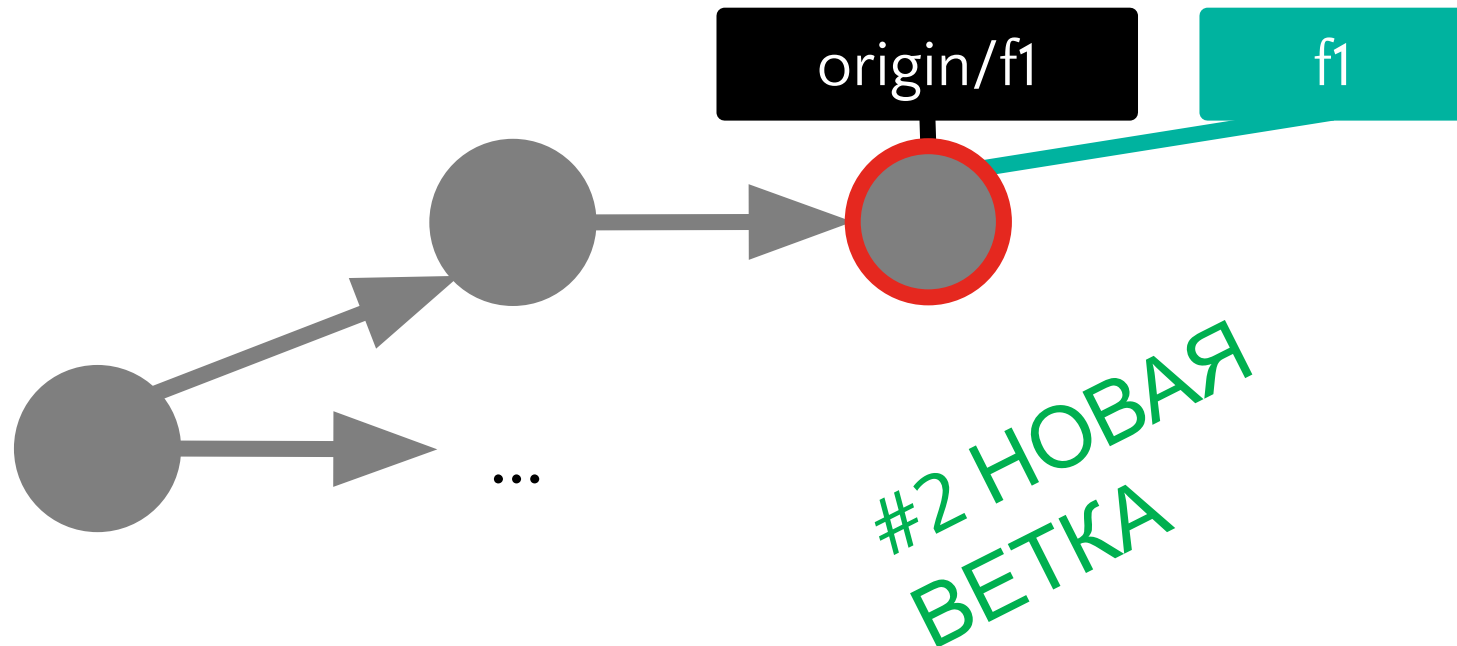
```
git checkout -b f1 origin/f1  
git checkout --track origin/f1
```

```
git checkout -b f1
```



# checkout upstream

---

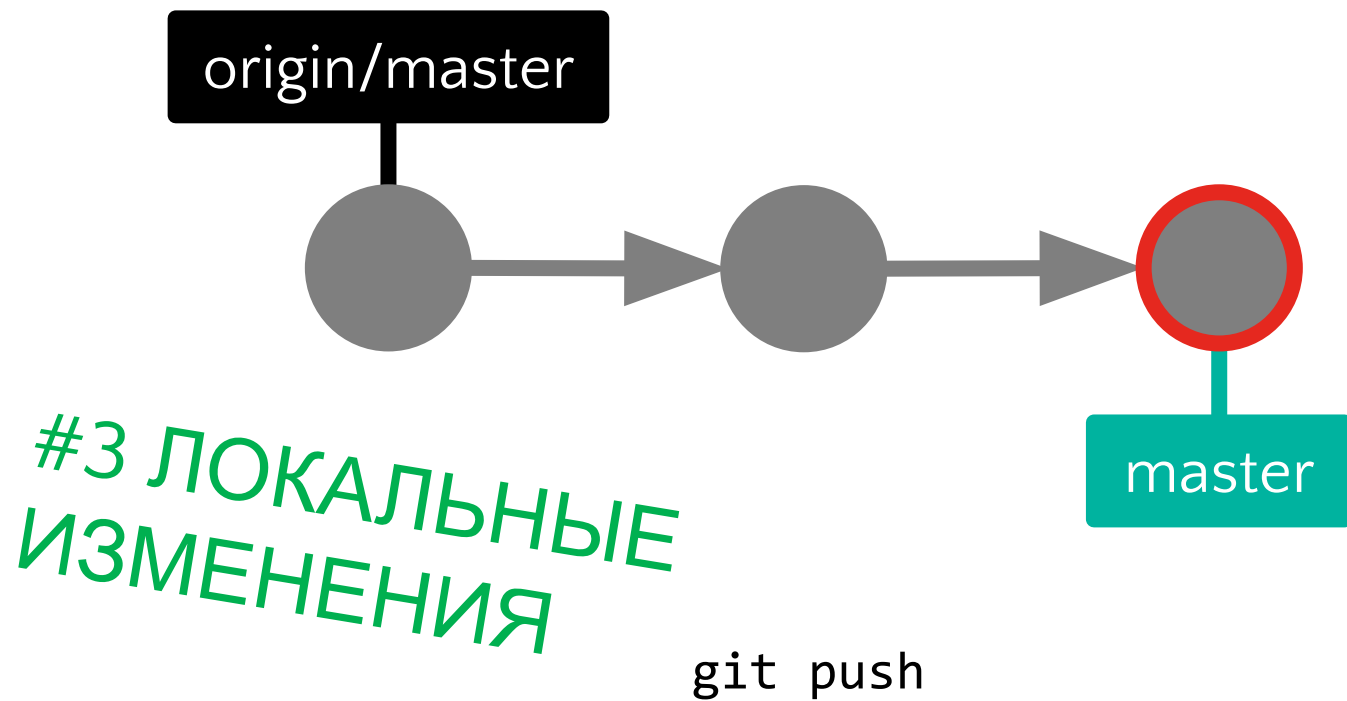


```
git checkout -b f1 origin/f1  
git checkout --track origin/f1
```

```
git checkout -b f1
```

# push

---

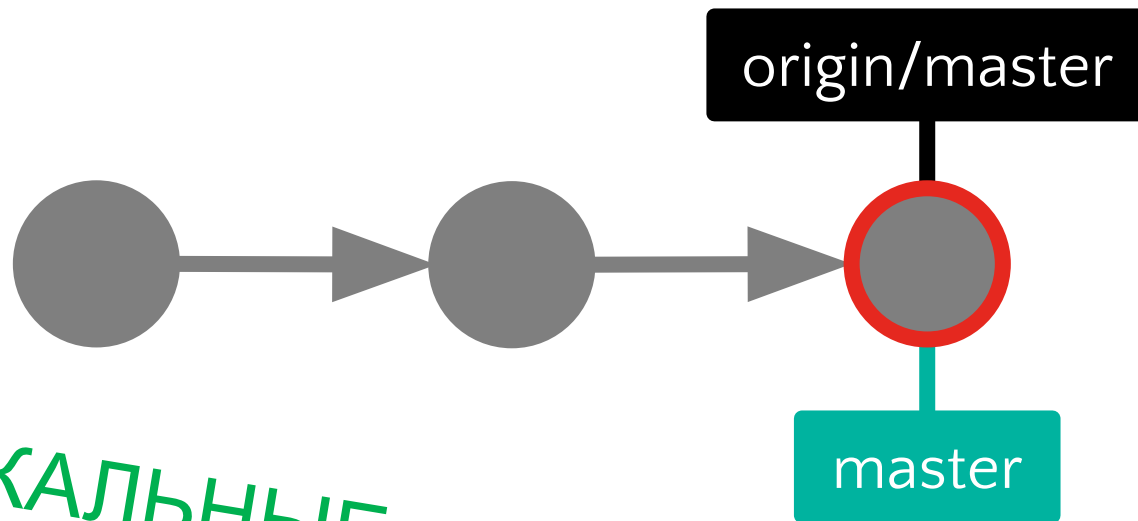


# push

---

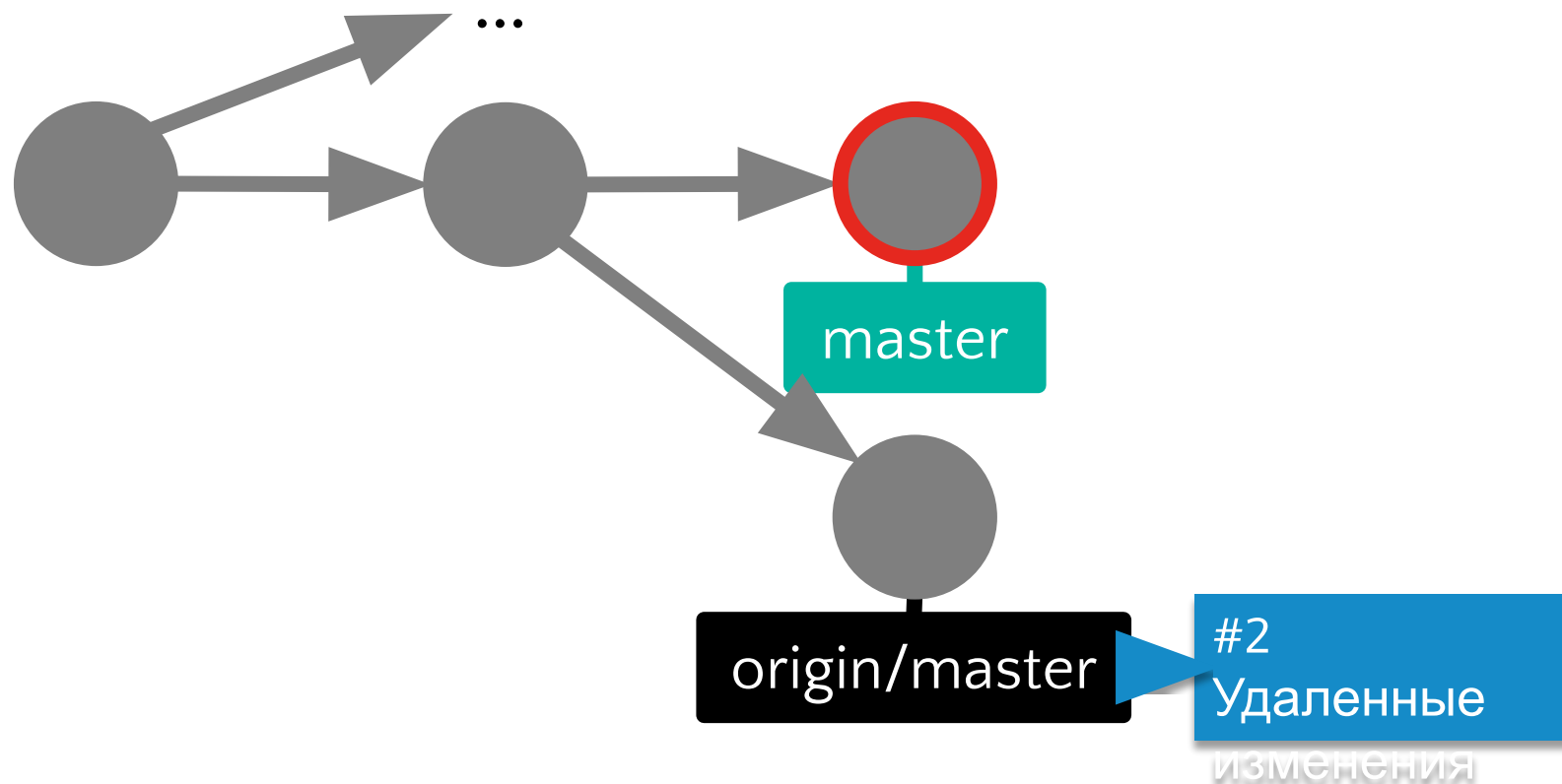
#3 ЛОКАЛЬНЫЕ  
ИЗМЕНЕНИЯ

git push



# Что делать?

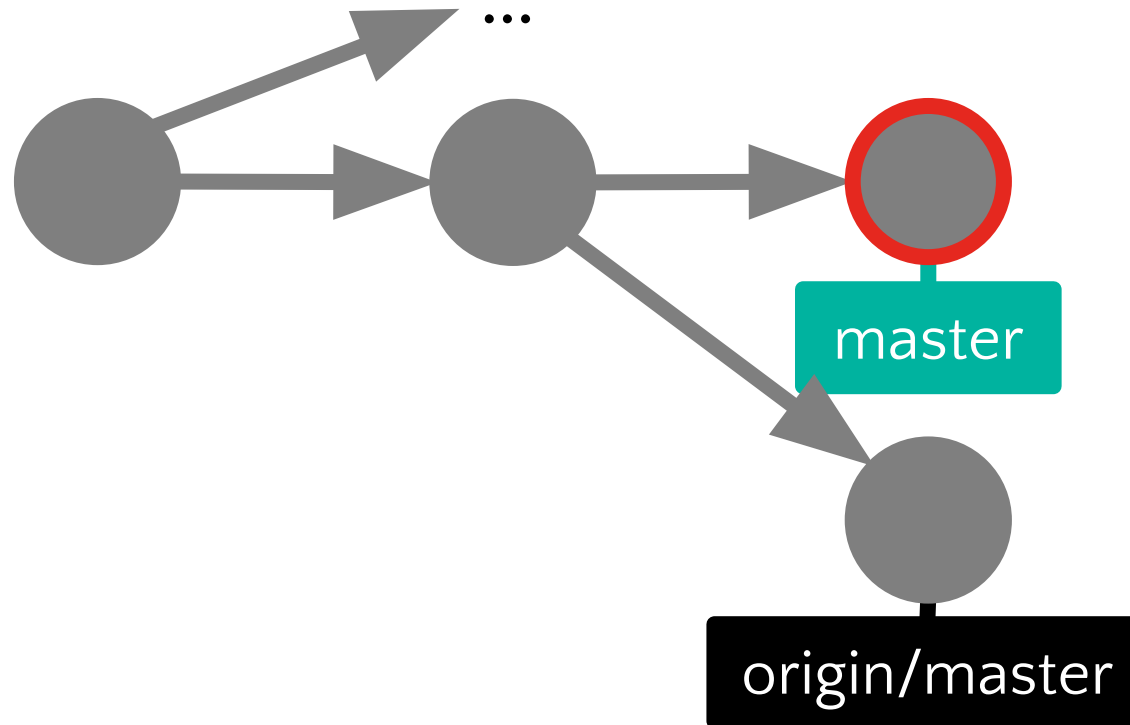
---



## #2 Удаленные изменения

---

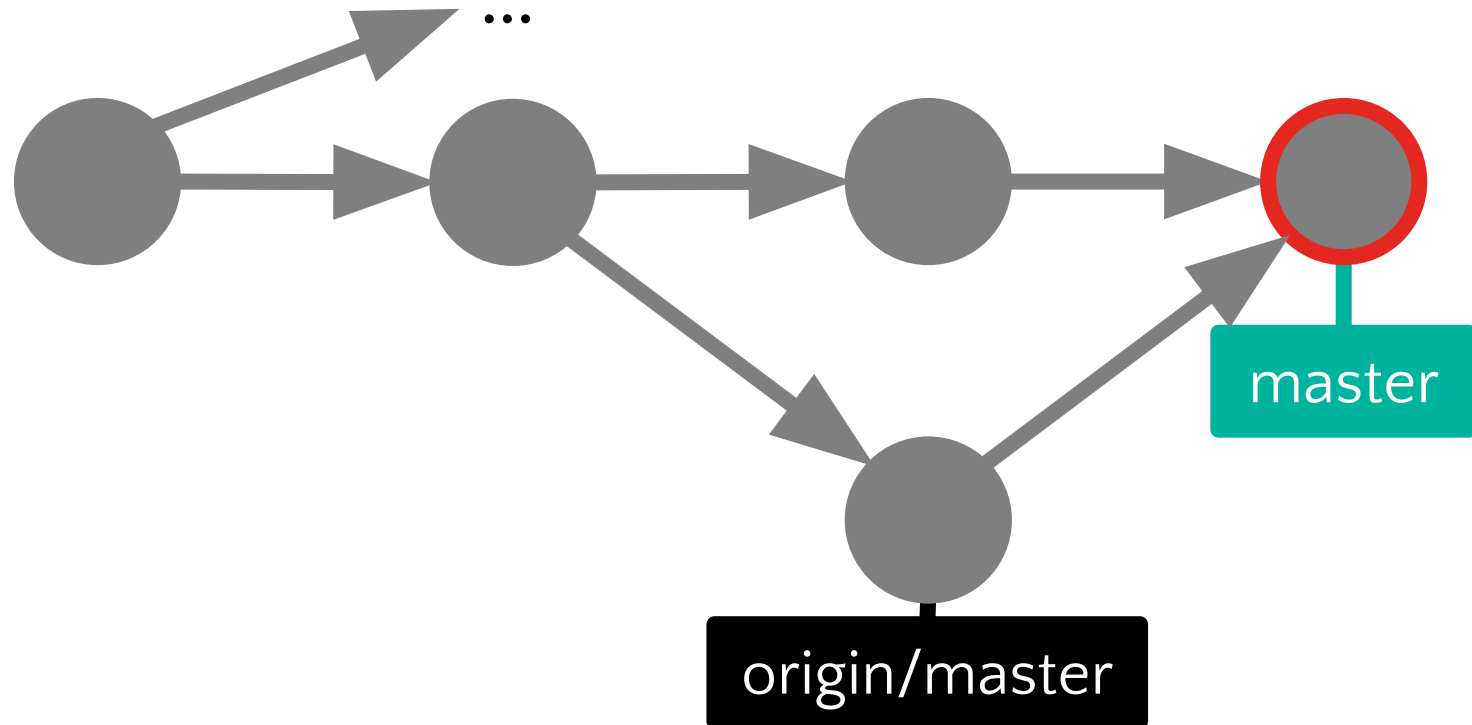
`git merge origin/master ?`



## #2 Удаленные изменения

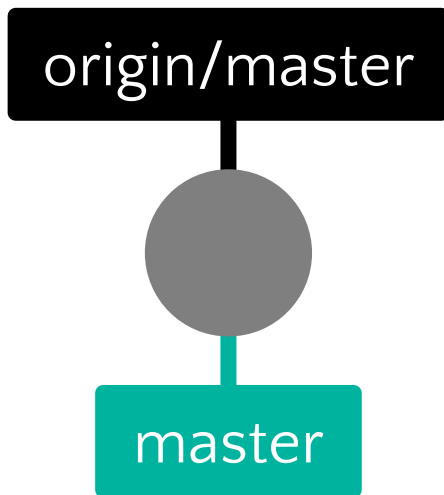
---

`git merge origin/master ?`



# pull

---

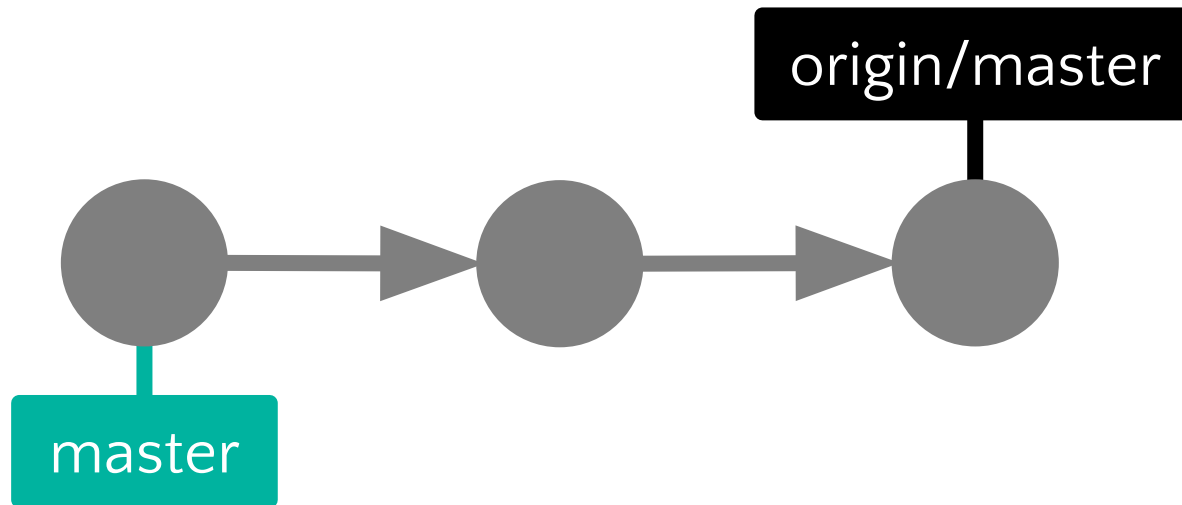


```
git pull origin =  
git fetch + git merge origin/<current_branch>
```

#2 УДАЛЕННО  
УЕХАЛО

# pull

---



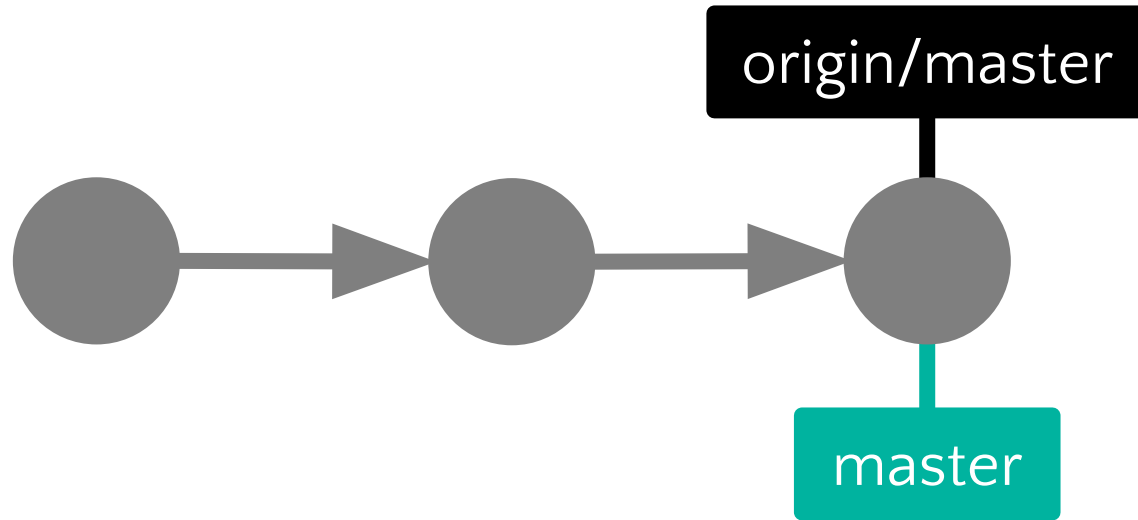
`git pull origin =`  
`git fetch + git merge origin/<current_branch>`

#2 УДАЛЕННО  
УЕХАЛО



# pull

---

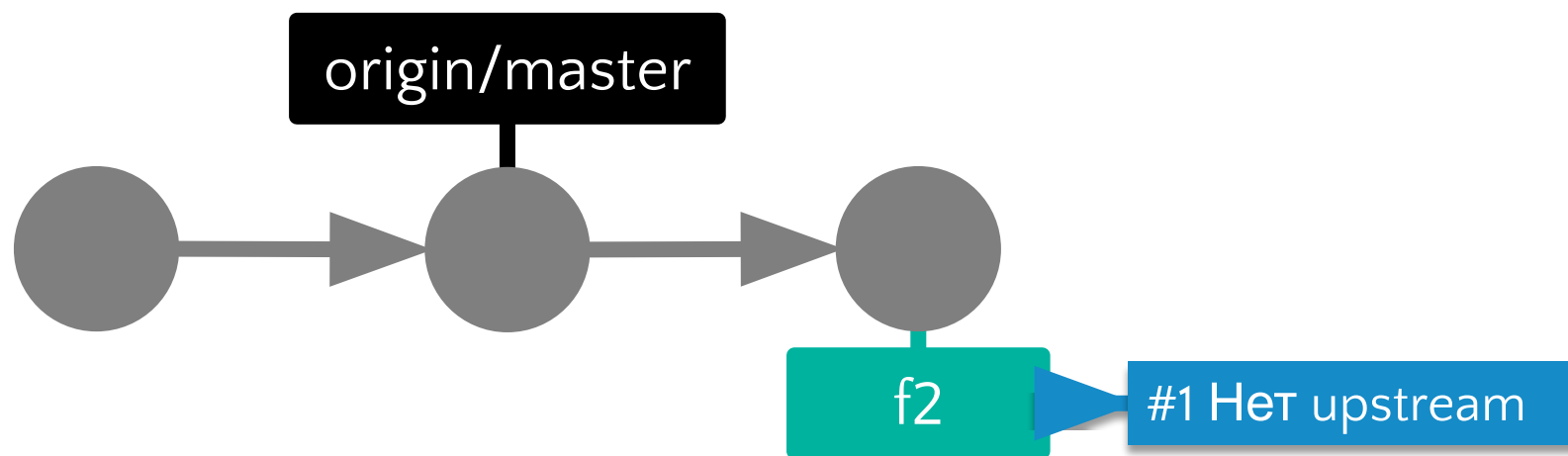


`git pull origin =`  
`git fetch + git merge origin/<current_branch>`

#2 УДАЛЕННО  
УЕХАЛО

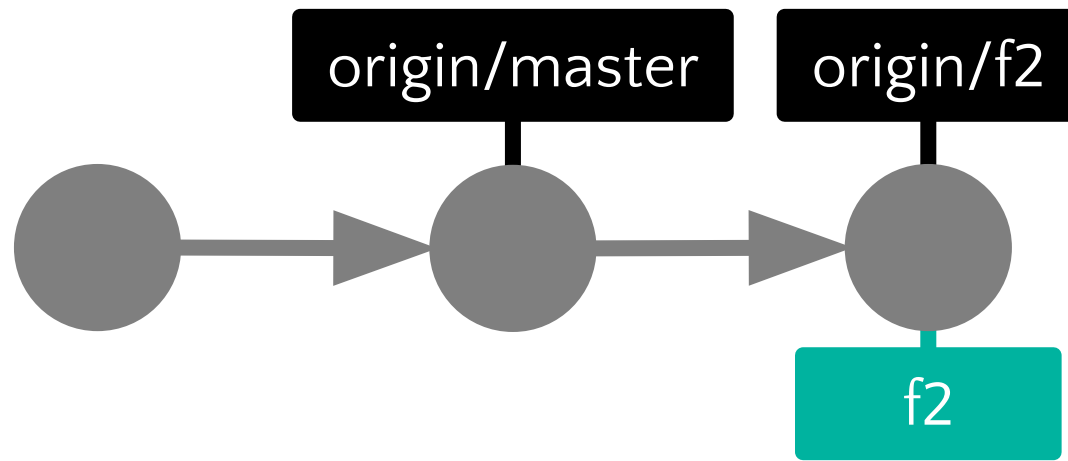
# Как начать фичу?

---



# #4 Her upstream

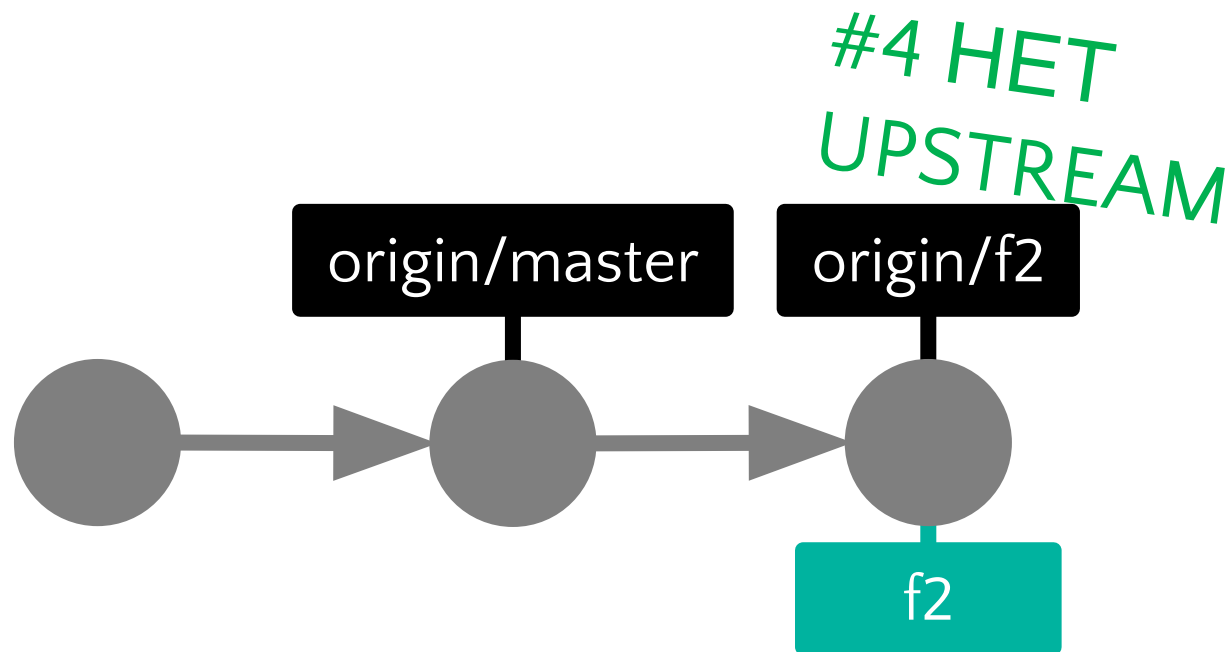
---



git push ?

# push ветки

---



```
git push <remote> <localBranch>:<remoteBranch>  
git push origin f2:f2  
git push -u origin HEAD
```

# Удаление ветки

---

## Локально

```
git branch -d <branchName>
```

## Удаленно

```
git push origin --delete <branchName>
```

```
git push origin :<branchName>
```

# Конфигурирование

---

# Конфигурирование

---

```
git config --system  
    %ProgramFiles%\Git\mingw64\etc\gitconfig
```

```
git config --global  
    %USERPROFILE%\gitconfig
```

```
git config  
    <repo>/git/config
```

# Настройка авторства

---

**.gitconfig**

[user]

name = username

email = username@mail.com

**КОМАНДЫ**

git config --global user.name Ivan Domashnikh

git config --global user.email domashnikh@mail.com



# Настройка для Windows

---

```
.gitconfig
```

```
[core]
```

```
autocrlf = true
```

```
safecrlf = true
```

## **КОМАНДЫ**

```
git config --global core.autocrlf true
```

```
git config --global core.safecrlf true
```

autocrlf – преобразование \r\n в \n

safecrlf – проверка обратимости преобразования \r\n в \n

# Выбор текстового редактора

---

```
.gitconfig
```

```
[core]
```

```
    editor = notepad
```

будет использоваться для ввода commit message

# Выбор merge tool

---

**.gitconfig**

[merge]

tool = TortoiseMerge

позволяет решать конфликты

# Разрешение конфликта в TortoiseMerge

The image shows the TortoiseMerge interface with three panes: 'Theirs', 'Mine', and 'Merged'.

**Theirs - style.css~other.u0x77n**

```
21 {
22 body {
23     background: blue;
24 }
25
26 h1 {
27     text-align: center;
28 }
```

**Mine - style.css.orig**

```
21 {
22 body {
23     background: green;
24 }
25
26 h1 {
27     text-align: center;
28 }
29
30 form {
31     text-align: center;
32 }
```

**\* Merged - style.css**

```
21 {
22 body {
23     background: green;
24 }
25
26 h1 {
27     text-align: center;
28 }
29
30 form {
31     text-align: center;
32 }
```

The conflict is resolved by taking the 'green' background from the 'Mine' version and the 'form' styling from the 'Mine' version, while keeping the 'h1' styling from the 'Theirs' version.

# Git Alias

---

## **.gitconfig**

[alias]

co = checkout

ci = commit

st = status

br = branch

hist = log --pretty=format:@"%h %ad | %s%d

[%an]" --graph --date=short

type = cat-file -t

dump = cat-file -p

undo = reset --hard HEAD~1

sm = submodule

# Bash Alias

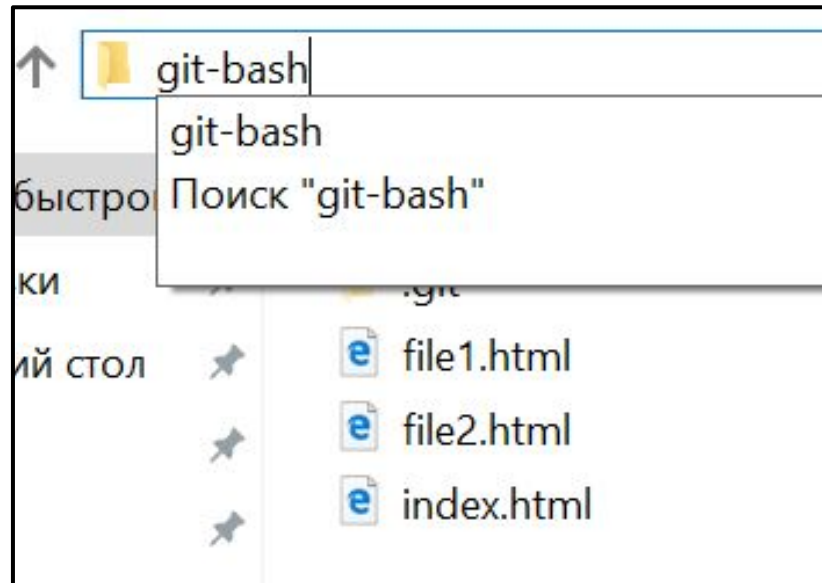
---

```
%USERPROFILE%\ .bashrc
alias less='less -r'
# --show-control-chars: help showing Korean or accented
characters
alias ls='ls -F --color --show-control-chars'
alias ll='ls -l'
alias gs='git status '
alias ga='git add '
alias gb='git branch '
alias gc='git commit'
alias gd='git diff'
alias go='git checkout '
alias gk='gitk --all&'
alias gx='gitx --all'
alias got='git '
alias get='git '
```

# Git Bash везде

---

Если добавить `C:\Program Files\Git` в `PATH`,  
то `git-bash` будет доступен через Win+R и адресную строку  
Проводника



# Игнорирование файлов

---

**.gitconfig**

[core]

excludesFile

**для репозитория**

<repo>/.git/info/exclude

**в любой папке и ее подпапках**

.gitignore



# OpenSSH

---

1. выполнить в Git Bash  
`ssh-keygen -t rsa -C "keyname"`
2. найти открытый ключ %USERPROFILE%\.ssh\id\_rsa.pub
3. Зарегистрировать открытый ключ на сервере

# PuTTY SSH

---

Git Extensions может использовать как PuTTY, так OpenSSH

1. открыть в Git Extensions **/Tools/PuTTY/Generate or import key**
2. сгенерировать ключ SSH-2 RSA, нажав **Generate**, а затем поведя мышкой вплоть до генерации ключа
3. открытый ключ появится прямо в диалоговом окне – его следует сохранить и зарегистрировать на сервере
4. на диск следует сохранить файл с расширением **.ppk**
5. теперь при выполнении **clone/fetch/push** можно нажать **Load SSH key** и выбрать для использования закрытый ключ **\*.ppk**

# Доверенные узлы

---

может требоваться доверие к удаленному репозиторию

список доверенных

`%USERPROFILE%\.ssh\known_hosts`

для добавления в доверенные

выполнить в Git Bash

**`ssh <host>`**

*e.g. `ssh github.com`*

для подтверждения соединения ввести

**`yes`**

# Напоследок

---

# amend

---

Можно добавить изменения в уже сделанный  
КОММИТ

```
git commit --amend
```

# reflog

---

Позволяет отобразить историю перемещений HEAD

Спасает, когда на нужный коммит не указывает ни тэг, ни ветка

```
git reflog
```

```
247ed61 HEAD@{0}: commit: Fix bug  
b7cad06 HEAD@{1}: checkout: moving from master to  
cool-feature  
b7cab06 HEAD@{2}: clone: from  
https://github.com/awesome-application
```

# clean

---

Удаление недобавленных в git файлов

```
git clean
```

# reset

soft

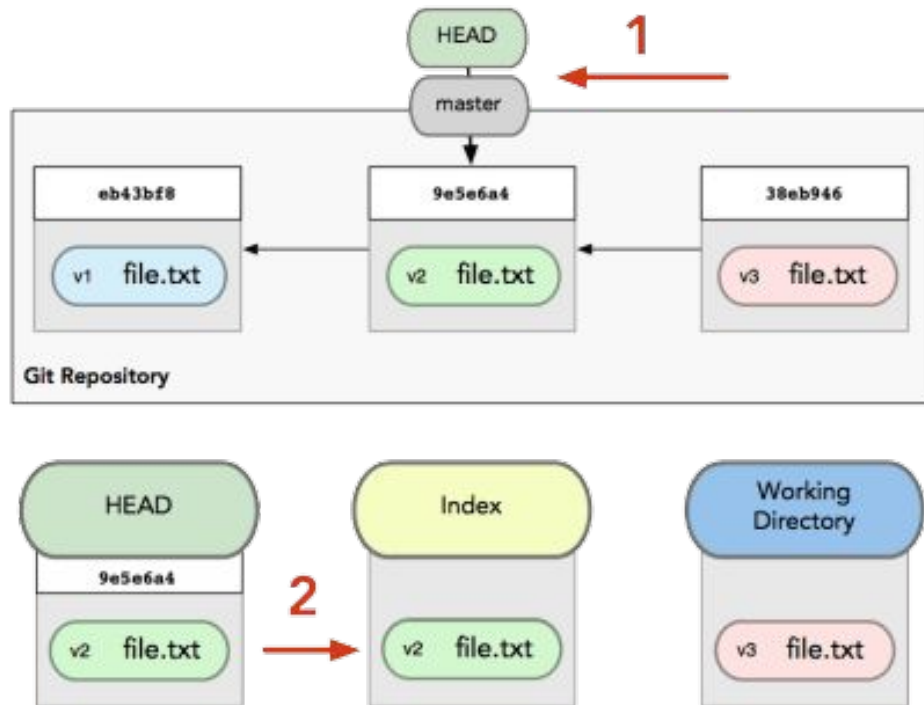
переносит HEAD и ветку

mixed

+ переписывает Index

hard

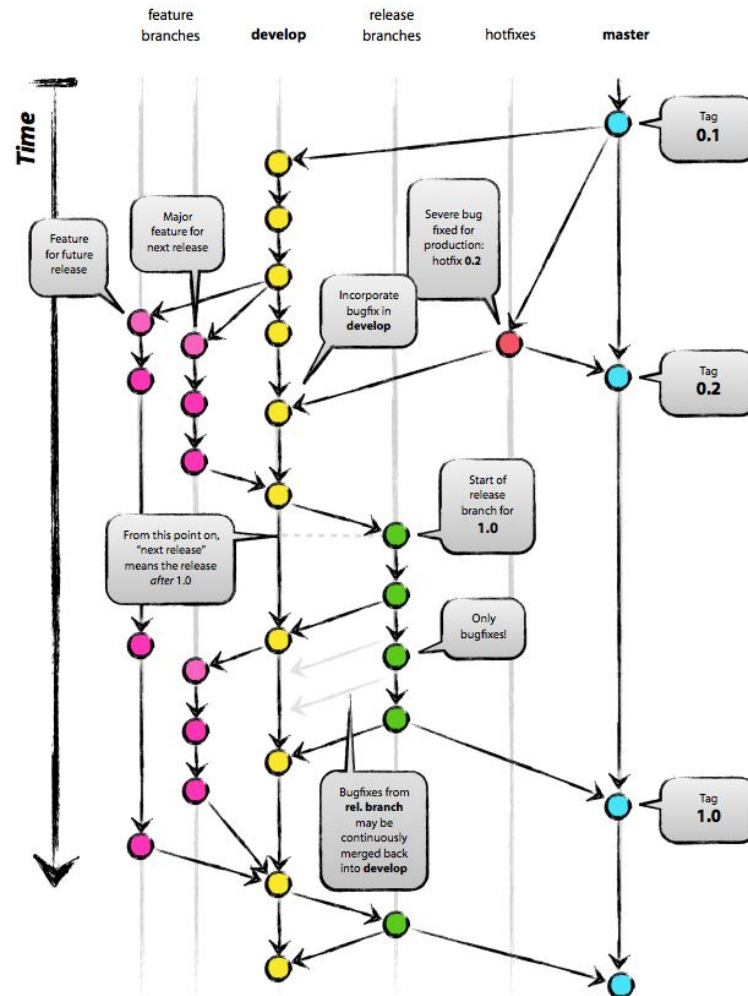
+ переписывает WD



**git reset [--mixed] HEAD~**



# flow



# ЧТИВО

---

**Классный практический курс на русском**

<https://githowto.com/ru>

**Про remote branches**

[Глава в Pro Git](#)

**Про reset**

<https://git-scm.com/blog/2011/07/11/reset.html>

**Основные фишки с картинками на английском**

<https://www.atlassian.com/pt/git/tutorial>

**Pro Git**

<https://git-scm.com/book/ru/v2>



# Кач

---

Git

<https://git-scm.com/>

Git Extensions

<https://sourceforge.net/projects/gitextensions/>

TortoiseGit

<https://tortoisegit.org/>

SourceTree

<https://www.sourcetreeapp.com/>



Вопросы?