

# Git

Артём Попцов

2015-10-24

**1** Ветвление

**2** Слияние

**3** Работа с удалёнными репозиториями

☞ Ветвь (англ. *branch*) – направление разработки, независимое от других.

Ветви создаются для:

- Параллельной работы над разными частями проекта
- Выпуска стабильных релизов проекта
- Тестирования “потенциально опасных” идей

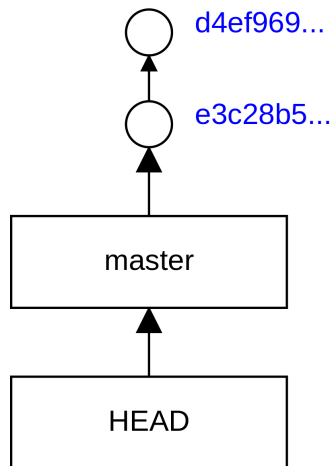


Мастер ветвления

# Создание ветви – 1

Создадим тестовый проект:

```
$ cd ~/src/my-project
$ git init
$ git add hello-world.txt
$ git commit -m "Initial commit"
... меняем hello-world.txt ...
$ git add hello-world.txt
$ git commit \
    -m "hello-world.txt: Update"
```



## Создание ветви – 2

Теперь создадим ветвь:

```
$ git branch branch-1
```

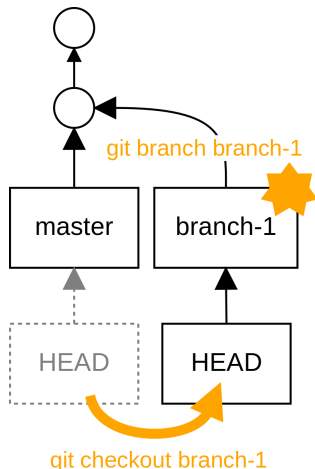
Переключимся на новую ветвь:

```
$ git checkout branch-1
```

...

Тоже, что описано выше, но одной командой:

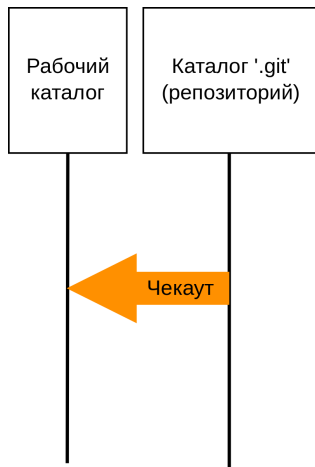
```
$ git checkout -b branch-1
```



# Как работает git checkout

`git checkout branch-1` делает две вещи:

- 1 Копирует в рабочий каталог файлы с верхушки ветви
- 2 Перемещает указатель HEAD на ветвь

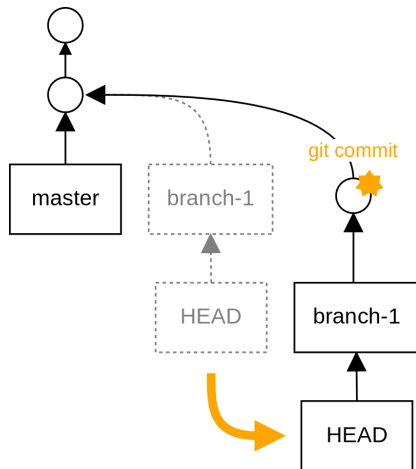


# Параллельная разработка – 1

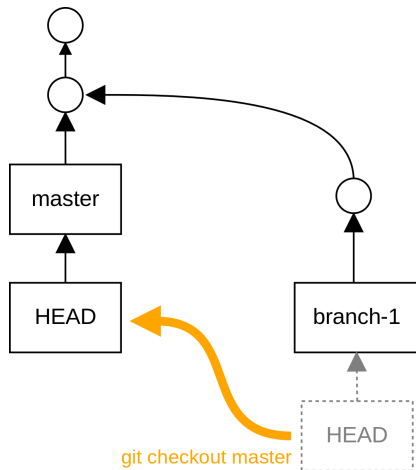
1 Меняем hello-world.txt

2 Фиксируем изменения:

```
$ git commit
```

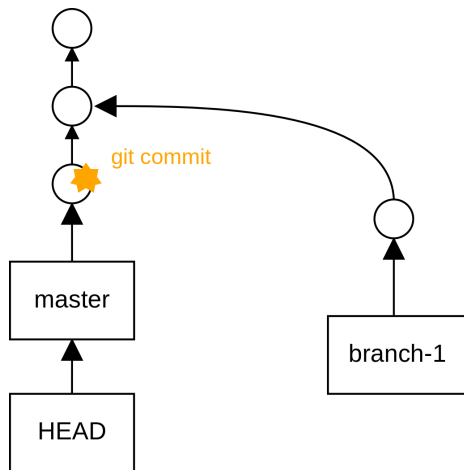


- 1 Меняем hello-world.txt
- 2 Фиксируем изменения:  
`$ git commit`
- 3 **Переходим на ветвь**  
master:  
`$ git checkout master`





- 1 Меняем hello-world.txt
- 2 Фиксируем изменения:  
`$ git commit`
- 3 Переходим на ветвь master:  
`$ git checkout master`
- 4 **Меняем** hello-world.txt
- 5 **Фиксируем** изменения:  
`$ git commit`



☞ Слияние (англ. *merge*) – объединение изменений из нескольких источников.

К примеру: слияние (“мёрж”) двух ветвей.

Типы слияния:

- fast-forward
- Остальные ;-)

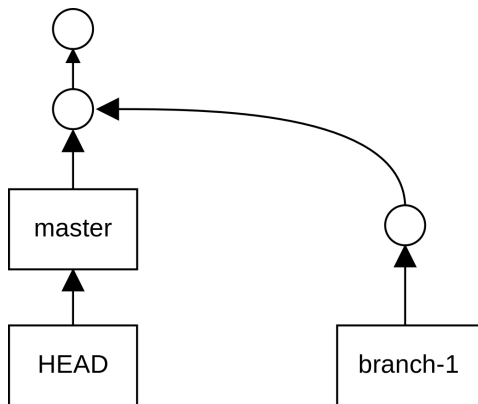


Ну что, мёржим?

# Простое слияние: fast-forward – 1

Начальное состояние:

- На branch-1 закоммичены (зафиксированы) изменения
- master не двигался с момента создания ветви branch-1



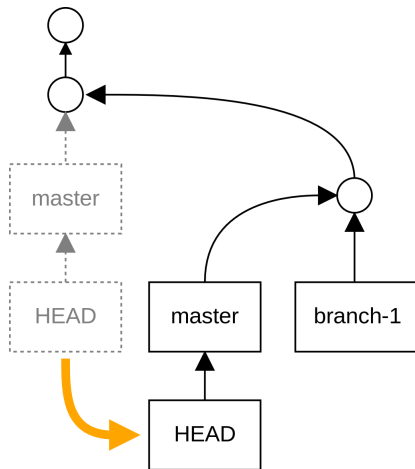
# Простое слияние: fast-forward – 2

```
$ git checkout master
```

```
$ git merge branch-1
```

Результат:

- Перемещается указатель на верхушку ветви
- Новых коммитов не создаётся

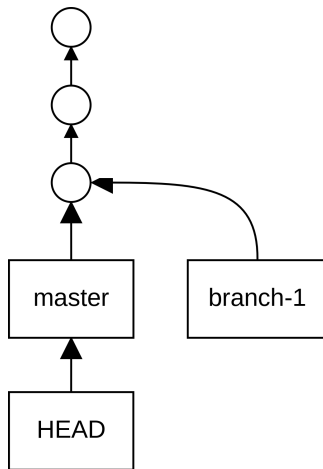


# Простое слияние: fast-forward – 3

```
$ git checkout master  
$ git merge branch-1
```

Результат:

- Перемещается указатель на верхушку ветви
- Новых коммитов не создаётся

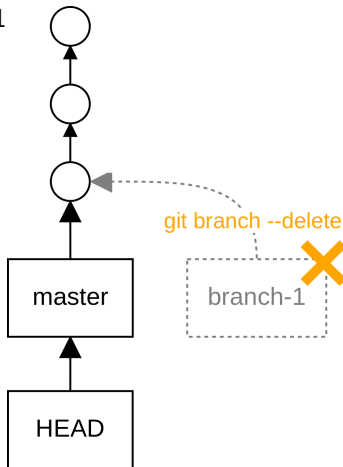
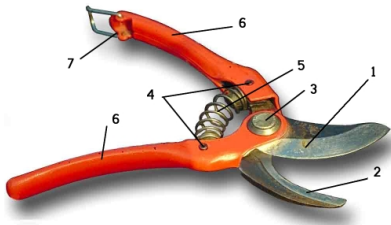


# Удаление ветви

```
$ git branch --delete branch-1
```

Результат:

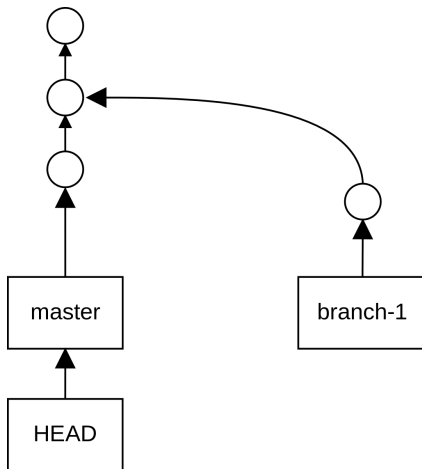
- Ветвь branch-1 удалена



# "Тяжёлый случай": recursive merge – 1

Начальное состояние:

- На branch-1  
закоммичены  
(зафиксированы)  
изменения
- На master  
закоммичены  
изменения после  
создания ветви  
branch-1
- Изменения на ветвях  
не пересекаются  
(сделаны в разных  
файлах)



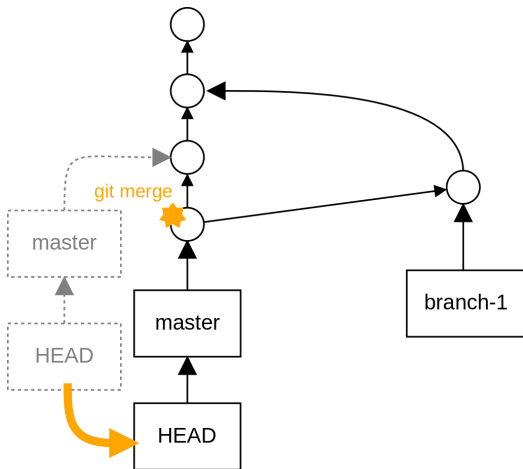
# "Тяжёлый случай": recursive merge – 2

```
$ git checkout master
```

```
$ git merge branch-1
```

Результат:

- Производится попытка автоматического слияния изменений
- Результатом слияния является новый коммит
- Перемещается указатель на вершку ветви



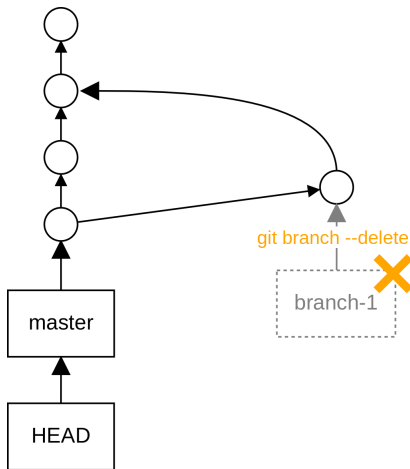


# "Тяжёлый случай": recursive merge – 3

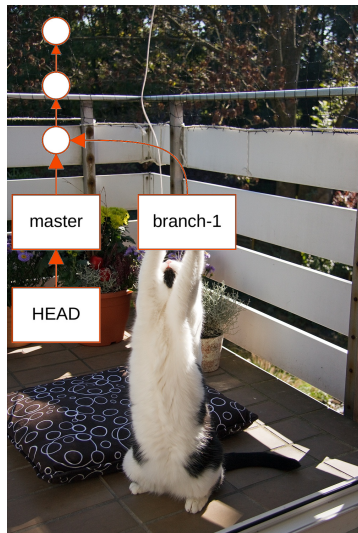
```
$ git branch \  
    --delete branch-1
```

Результат:

- Ветвь branch-1 удалена



- Создать ветвь и переключиться на неё:  
`$ git checkout -b branch-1`
- Замёрзить ветвь branch-1 на master:  
`$ git checkout master`  
`$ git merge branch-1`
- Удалить ветвь branch-1:  
`$ git branch --delete branch-1`
- Показать список ветвей:  
`$ git branch`



# Разрешение конфликтов – 1

```
$ git init
```

```
$ echo "hello" > hello-world.txt
```

```
$ git add hello-world.txt
```

```
$ git commit
```

```
$ git checkout -b branch-1
```

```
$ echo "world" >> hello-world.txt
```

```
$ git add hello-world.txt
```

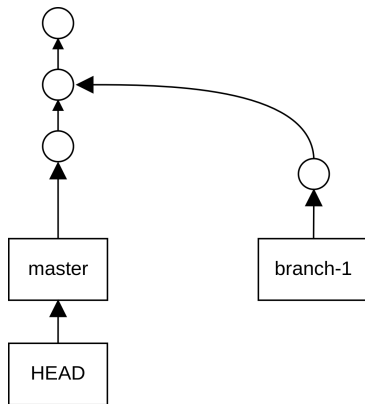
```
$ git commit
```

```
$ git checkout master
```

```
$ echo "git" >> hello-world.txt
```

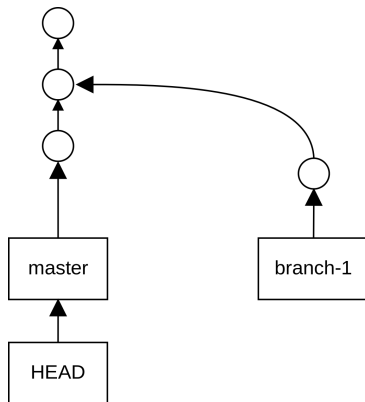
```
$ git add hello-world.txt
```

```
$ git commit
```



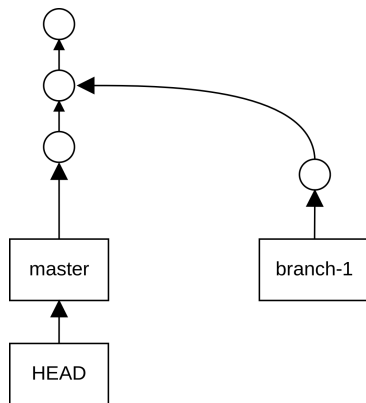
Начальное состояние:

- На `branch-1` закоммичены (зафиксированы) изменения
- На `master` закоммичены изменения после создания ветви `branch-1`
- Изменения на ветвях пересекаются (изменены одни и те же файлы)



# Разрешение конфликтов – 3

```
$ git merge branch-1
Auto-merging hello-world.txt
CONFLICT (content): Merge conflict
in hello-world.txt
Automatic merge failed;
fix conflicts and then commit
the result.
```



Статус репозитория:

```
$ git status
```

```
On branch master
```

```
You have unmerged paths.
```

```
  (fix conflicts and run "git commit")
```

```
Unmerged paths:
```

```
  (use "git add <file>..." to mark resolution)
```

```
both modified:   hello-world.txt
```

```
no changes added to commit (use "git add"  
and/or "git commit -a")
```

Файл, в котором возник конфликт, с метками Git:

```
$ cat hello-world.txt
hello
<<<<<<< HEAD
git
=====
world
>>>>>>> branch-1
```

Разрешаем конфликт вручную, редактируя файл. Сохраняем.  
Результат:

```
$ cat hello-world.txt
hello
git
world
```

# Разрешение конфликтов – 6

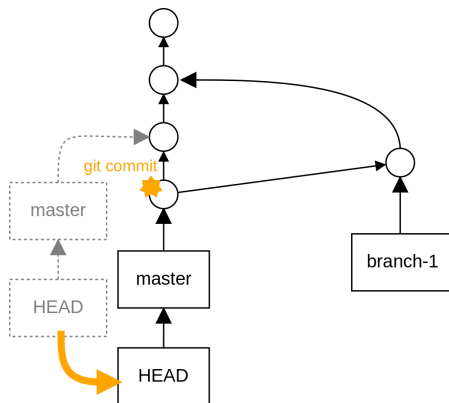
Фиксируем результат слияния:

```
$ git add hello-world.txt  
$ git commit
```


Результат:

```
$ git log -1  
commit ea835e9...  
Merge: 35abf34 f3f5057  
Author: Artyom V. Poptsov ...  
Date: Fri Aug 26 ...
```

Merge branch 'branch-1'

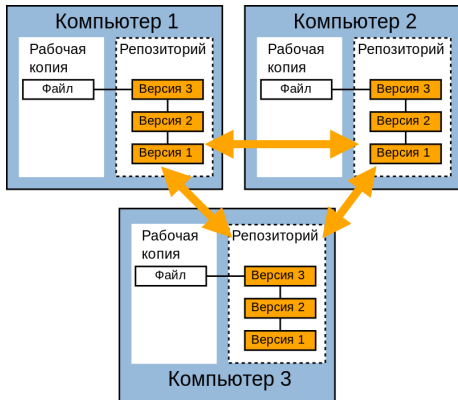




 **Git – Распределённая**  
система контроля версий.

Следовательно:

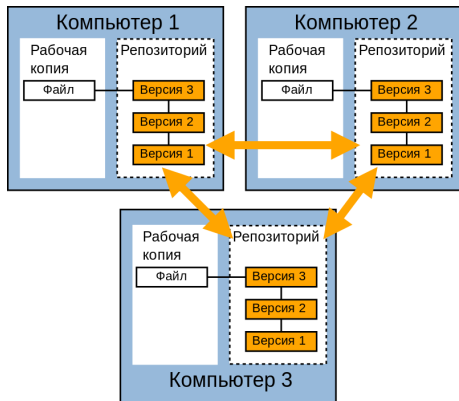
- В большинстве случаев локально хранится полная копия репозитория.
- Копии репозитория могут быть синхронизированы между собой.

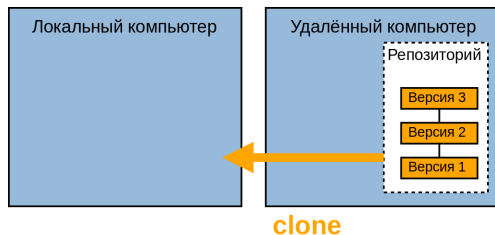


# Работа с удалёнными репозиториями – 2

Основные команды:

- **clone** – Получение копии репозитория.
- **pull** – Получение изменений из удалённого репозитория.
- **push** – Выгрузка изменений в удалённый репозиторий.





Получение копии (клонирование) репозитория:

```
$ git clone https://example.com/gitproject.git
```

Другие способы клонирования:

```
$ git clone /opt/git/gitproject.git
```

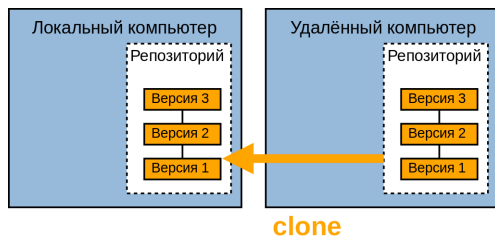
```
$ git clone ssh://user@example.com/gitproject.git
```

```
$ git clone git://example.com/gitproject.git
```

...

👉 При клонировании удалённый репозиторий будет запомнен в локальной копии под именем `origin`.

## git clone - 2



Получение копии (клонирование) репозитория:

```
$ git clone https://example.com/gitproject.git
```

Другие способы клонирования:

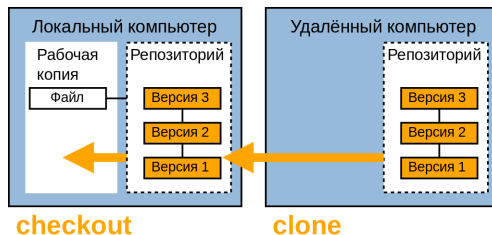
```
$ git clone /opt/git/gitproject.git
```

```
$ git clone ssh://user@example.com/gitproject.git
```

```
$ git clone git://example.com/gitproject.git
```

...

👉 При клонировании удалённый репозиторий будет запомнен в локальной копии под именем `origin`.



Получение копии (клонирование) репозитория:

```
$ git clone https://example.com/gitproject.git
```

Другие способы клонирования:

```
$ git clone /opt/git/gitproject.git
```

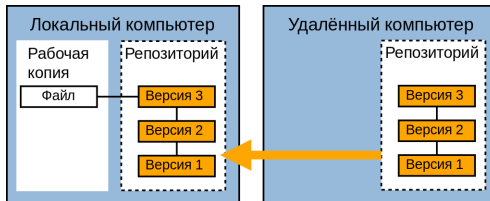
```
$ git clone ssh://user@example.com/gitproject.git
```

```
$ git clone git://example.com/gitproject.git
```

...

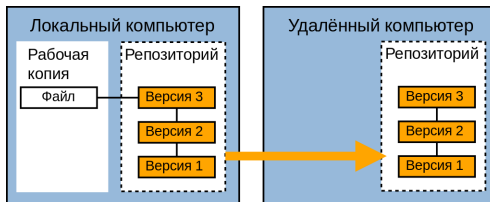
👉 При клонировании удалённый репозиторий будет запомнен в локальной копии под именем `origin`.

# git pull



Получение новых изменений с ветви `master` из удалённого репозитория `origin`:

```
$ git pull origin master
```



Выгрузка изменений с ветви master в удалённый репозиторий origin:

```
$ git push origin master
```

Выгрузка изменений с **текущей** ветви в удалённый репозиторий origin:

```
$ git push origin HEAD
```

👉 Ни один котёнок не пострадал при подготовке этой презентации!

Эл. почта: `poptsov.artiom@gmail.com`

Презентация и её “исходники” под лицензией Creative Commons:

`github.com/artiom-poptsov/talks/tree/master/vcs`

Спасибо, что выслушали :-)

## Вопросы?



Copyright ©2015 Artyom V. Poptsov  
<poptsov.artiom@gmail.com>

Права на копирование других изображений, использованных в данной работе, принадлежат их владельцам.

Данная работа распространяется на условиях лицензии Creative Commons Attribution-ShareAlike 4.0 International:  
<https://creativecommons.org/licenses/by-sa/4.0/>