

# Введение в системы управления версиями

Артём Попцов

2015-10-20

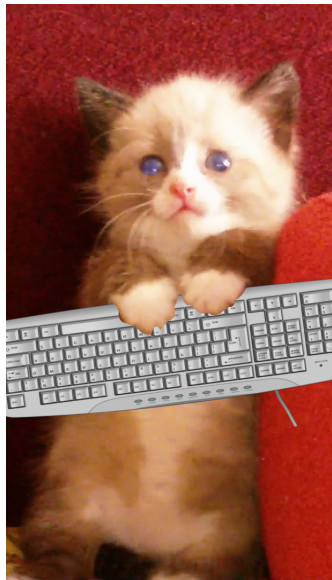
- 1 Задачи, решаемые системами управления версиями
- 2 Виды систем управления версиями
- 3 Система управления версиями Git
- 4 Заключение

- Начало работы
  - курсовая.tex
- Прошла неделя
  - курсовая.tex, курсовая.tex.bak
- Ещё некоторое время спустя
  - курсовая.tex, курсовая.tex.bak, курсовая-old.tex
- Показали преподавателю
  - курсовая.tex, курсовая.tex.bak, курсовая-old.tex, курсовая-new.tex
- Внесли изменения
  - курсовая.tex, курсовая.tex.bak, курсовая-old.tex, курсовая-old2.tex, курсовая-final.tex
- ...

# В ходе работы возникают вопросы...

- Какая версия последняя (актуальная)?
- Где находится актуальная версия? Она на флэшке?
- В чём различия между старой и новой версией?
- Я ничего не помню. Что я делал вчера?

**Ой!** Я случайно  
перезаписал новую  
версию старой.  
Что делать?..



Я должен использовать систему управления версиями!



Я прозрел!

# Что такое “система управления версиями”?

👉 Система управления версиями (англ. *Version Control System*, сокр. *VCS*) – ПО для облегчения работы с изменяющейся информацией.

Основные возможности систем управления версиями:

- **Обратимость** – возможность вернуться к предыдущему состоянию.
- **Согласованность** – возможность совместной работы с одними и теми же данными в одно и то же время.
- **Аннотирование** – возможность сохранять метаданные об изменениях, комментарии от автора изменений.

Базовые операции:

- Получение рабочей копии файлов из репозитория.
- Запись изменений в репозиторий.
- Просмотр истории файлов.

Как самостоятельное приложение:

- Разработка ПО.
- Web-разработка.
- Научные работы, книги, ...
- Векторная графика.
- Хранение конфигурационных файлов.
- ...

В составе других приложений:

- Wikipedia и другие wiki.
- Системы документооборота (например, Alfresco.)
- “Облачные” сервисы (Google Docs, файловые хранилища.)
- Текстовые процессоры.
- ...



- Репозиторий (англ. *repository*)
- “Чекаут” (англ. *check out*, сокр. *co*)
- Рабочая копия (англ. *working copy*)
- Различие (англ. *change, diff, delta*)
- Коммит, “чек-ин” (англ. *commit, check-in*, сокр. *ci*)
- Коммиттер (англ. *committer*)
- Ветвь (англ. *branch*)
- Ветвление (англ. *branching*)
- Верхушка, или “голова” ветви (англ. *tip, head*)
- Слияние, мёрж (англ. *merge*)
- Конфликт (англ. *conflict, merge conflict*)
- Метка, тэг (англ. *label, tag*)



По расположению репозитория:

- **Локальные** системы управления версиями (Local VCS)
- **Централизованные** (клиент-серверные) системы управления версиями (Centralized VCS)
- **Распределённые** системы управления версиями (Distributed VCS)

По способу работы с файлами:

- Оперирующие отдельными файлами
- Оперирующие наборами файлов

По способу разрешения конфликтов:

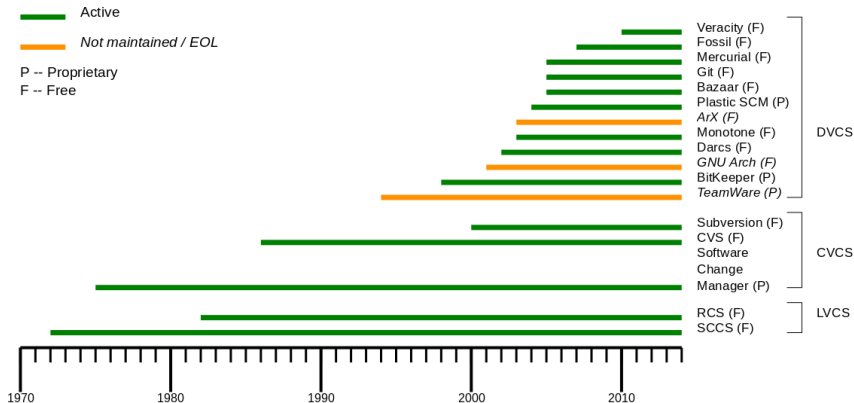
- Блокировка
- Слияние изменений
  - Перед коммитом
  - После коммита

По способу сохранения истории:

- Различия между версиями (дельты)
- Слелки состояния (снэлпшоты)

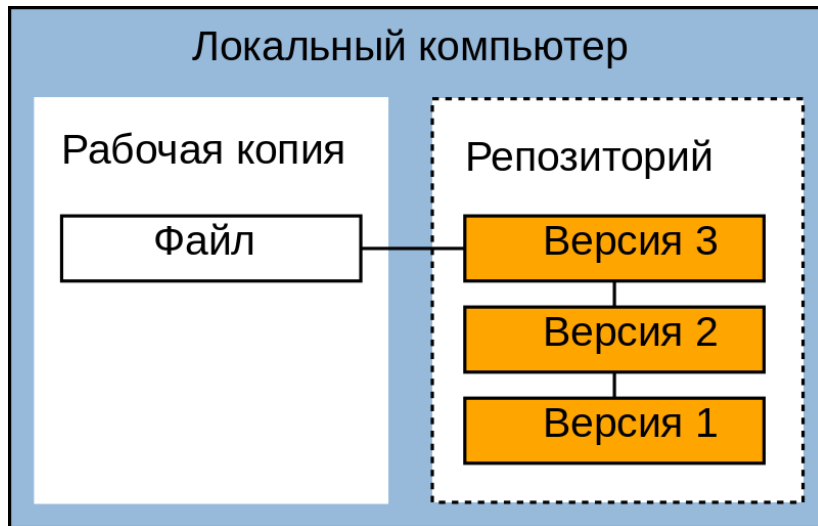
...

# История систем управления версиями





Какую VCS выбрать?



# Revision Control System (RCS)

Сайт: [gnu.org/s/rcs](http://gnu.org/s/rcs)

Некоторые факты:

- Создана примерно в 1982-м году.
- Локальная, оперирует отдельными файлами, использует блокировку для предотвращения конфликтов.

Преимущества:

- Проста в использовании.

Недостатки:

- Работа с ветками может быть нетривиальной.
- Каждый файл отслеживается отдельно.
- Не предоставляет контроль целостности.
- Не позволяет удалять файлы.
- Нет поддержки переименования файлов.
- Нет поддержки проектов с несколькими каталогами.
- ...

# Пример использования RCS

- 1 Переходим в каталог с проектом:

```
$ cd ~/src/my-project
```

- 2 Создаём служебный каталог RCS:

```
$ mkdir RCS
```

- 3 "Чекиним"(коммитим) новый файл:

```
$ ci hello-world.txt
```

При чекине RCS предложит ввести описание коммита.

- 4 "Чекаутим" файл, с блокировкой для редактирования:

```
$ co -l hello-world.txt
```

- 5 Редактируем файл ...

- 6 Чекиним изменения:

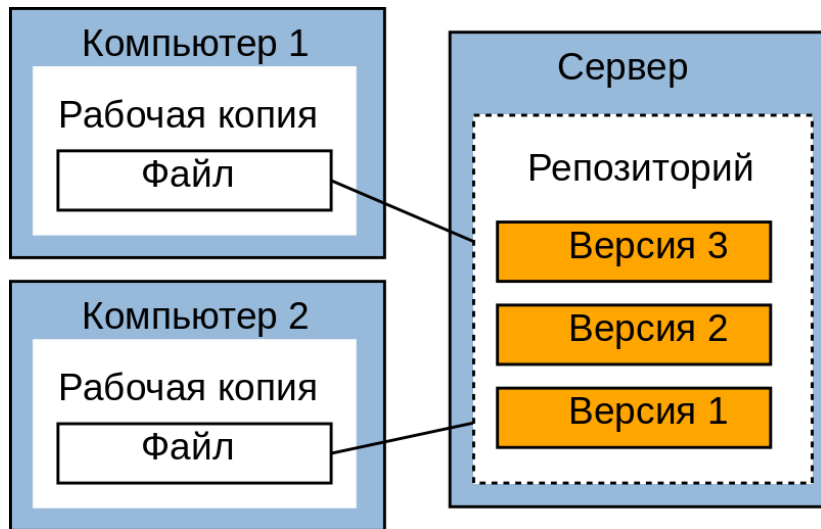
```
$ ci -l hello-world.txt
```

- 7 Смотрим лог:

```
$ rlog hello-world.txt
```



# Централизованные системы управления версиями



# Subversion (SVN)

Сайт: [subversion.apache.org](http://subversion.apache.org)

Некоторые факты:

- Создана в 2000-м году.
- Централизованная, оперирует наборами файлов, использует слияние изменений для разрешения конфликтов.

Преимущества:

- Атомарные коммиты.
- Возможность переименования, копирования, перемещения и удаления файлов с сохранением истории.
- Работа с каталогами, символическими ссылками.
- Поддержка бинарных файлов.
- Позволяет легко увидеть общую картину – кто и над чем работает.
- Предоставляет администратору полный контроль над доступом к репозиторию.

# Централизованные VCS ограничивают вашу свободу!



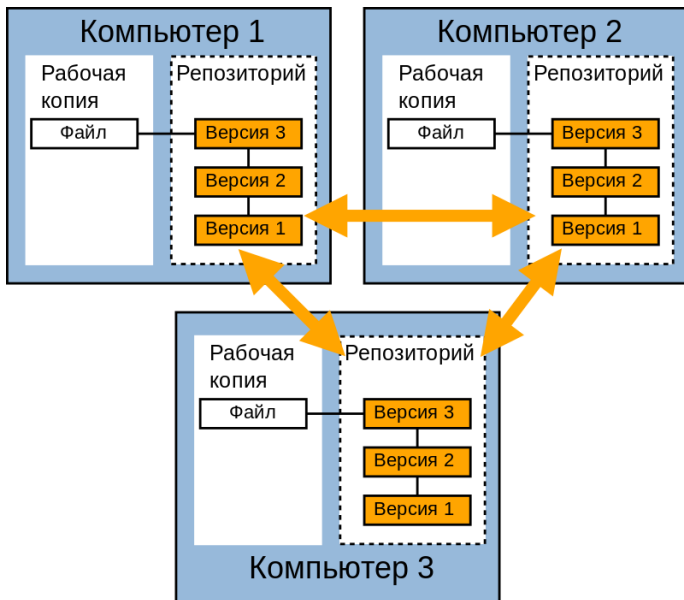
В нашем проекте используют централизованную VCS.



В нашем проекте используют ClearCase.

- В случае недоступности сервера вы ограничены в действиях.
- Скорость выполнения операций зависит от скорости подключения к серверу.
- Единая точка отказа.
- Централизация диктует способ организации работы команды.

# Распределённые системы управления версиями



# Преимущества распределённых VCS

- Нет зависимости от одного сервера, каждый клон является полноценным репозиторием  $\Rightarrow$  нет единой точки отказа
- Большая часть операций локальны  $\Rightarrow$  высокая скорость работы, нет зависимости от подключения к сети
- Взаимодействие между разработчиками организовано по принципу peer-to-peer  $\Rightarrow$  возможна адаптация различных моделей разработки
- Не нужно поднимать сервер  $\Rightarrow$  проще начать работу

Все преимущества локальных VCS плюс дополнительные “плюшки” современных VCS. Бесплатно!

Bay!

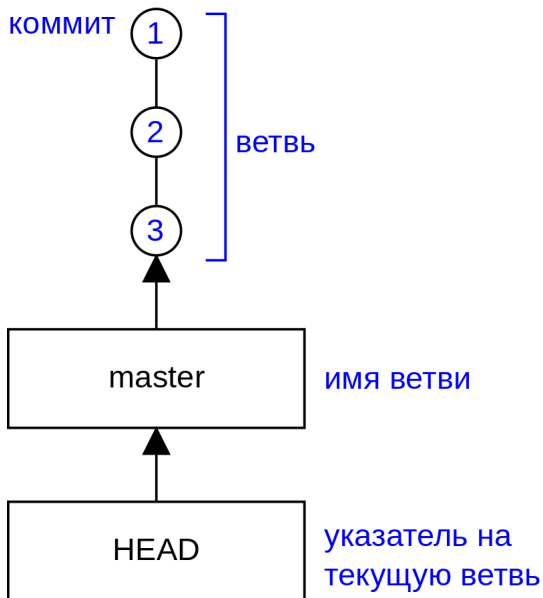


Bay!

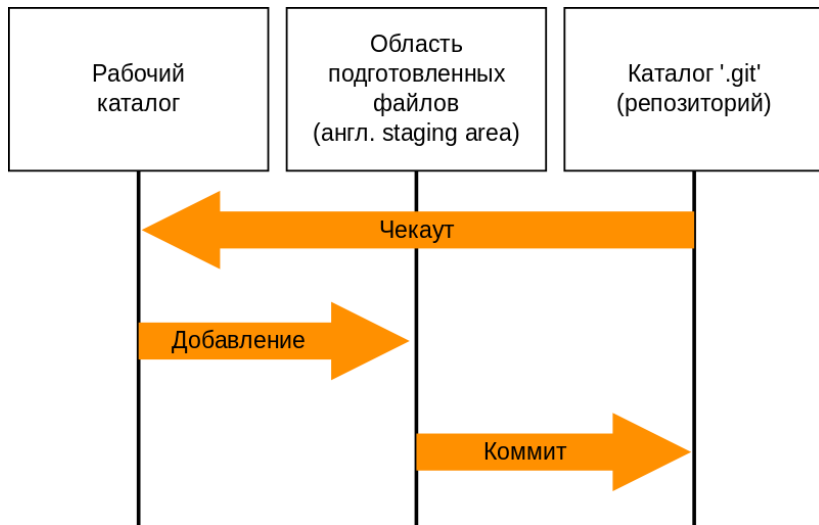


- Сайт: [git-scm.com](https://git-scm.com)
- Создан в 2005-м году; история Git связана с историей ядра Linux.
- Распределённый, оперирует наборами файлов, хранит слепки состояния.
- Следит за целостностью данных.
- Большая часть операций локальны.
- Поддержка нелинейной разработки, лёгкость создания веток и работы с ними.
- Возможность использования различных моделей организации работы команды.
- Эффективная работа с большими проектами.





# Локальные операции



## Задаём основные настройки:

```
$ git config --global user.name "Vasily I. Pupkin"
$ git config --global user.email "vip@example.ru"
```

## Смотрим пользовательский файл настроек:

```
$ cat ~/.gitconfig
[user]
    name = Vasily I. Pupkin
    email = vip@example.ru
```

## Получение справки:

```
$ git --help          # Список часто используемых команд
$ git config -h        # Краткая справка
$ git config --help    # Подробная справка (man-страница)
$ man git-config       # man-страница команды
```

- 1 Переходим в каталог с проектом:  
`$ cd ~/src/my-project`
- 2 Инициализируем репозиторий:  
`$ git init`
- 3 Подготавливаем файл к коммиту:  
`$ git add hello-world.txt`
- 4 Коммитим изменения:  
`$ git commit -m "Initial commit"`
- 5 Меняем файл `hello-world.txt` ...
- 6 Подготавливаем изменения к коммиту:  
`$ git add hello-world.txt`
- 7 Коммитим изменения:  
`$ git commit -m "hello-world.txt: Update"`
- 8 Смотрим историю изменений:  
`$ git log`

## Начальное состояние каталога с проектом:

```
$ cd ~/src/my-project/  
$ ls -A  
hello-world.txt  
$ cat hello-world.txt  
hello
```

## Инициализируем пустой репозиторий:

```
$ git init  
Инициализирован пустой репозиторий Git  
в /home/vip/src/my-project/.git/
```

## Что в каталоге?

```
$ ls -A  
.git  hello-world.txt
```

Смотрим, что получилось:

```
$ git status
```

На ветке master

Начальный коммит

Неотслеживаемые файлы:

(используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)

hello-world.txt

ничего не добавлено в коммит, но есть неотслеживаемые файлы (используйте "git add", чтобы отслеживать их)

Добавляем новый файл:

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

Смотрим на результат:

```
$ git status
```

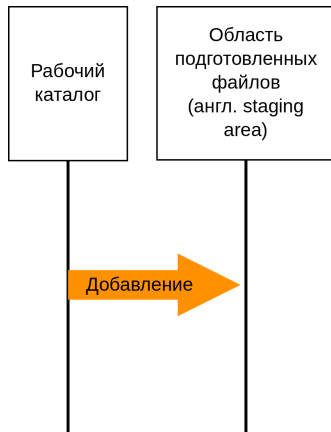
На ветке master

Начальный коммит

Изменения, которые будут  
включены в коммит:

(используйте  
«`git rm --cached <файл>...`»,  
чтобы убрать из индекса)

новый файл:      `hello-world.txt`



## Коммитим изменения:

```
$ git commit -m "Initial commit"
[master (корневой коммит) d4ef969]
  Initial commit
  1 file changed, 1 insertion(+)
  create mode 100644 hello-world.txt
```

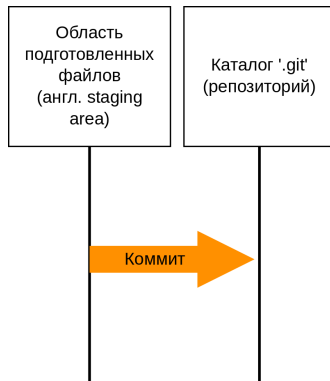
## Смотрим на результат:

```
$ git status
```

На ветке master  
нечего коммитить, нет изменений  
в рабочем каталоге

## Что в каталоге?

```
$ ls -A
.git  hello-world.txt
```





**Делаем изменения:**

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

**Смотрим различия:**

```
$ git diff
diff --git a/hello-world.txt b/hello-world.txt
index ce01362..94954ab 100644
--- a/hello-world.txt
+++ b/hello-world.txt
@@ -1,2 @@
    hello
+world
```

**Коммитим изменения:**

```
$ git add hello-world.txt
$ git commit -m "hello-world.txt: Update"
```

## Смотрим лог изменений:

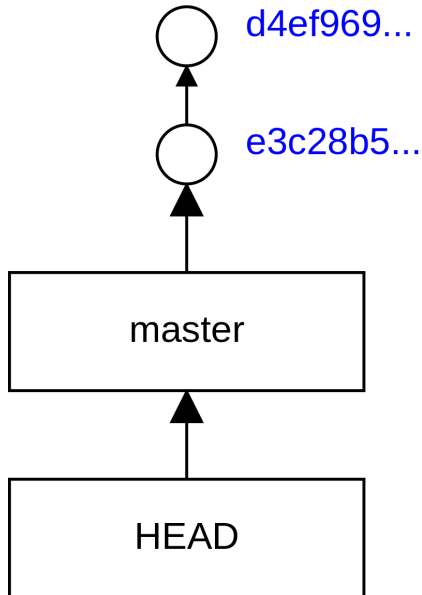
```
$ git log
commit e3c28b5608ef0a2ab9f042d9633c4d6e1a5fc419
Author: Vasily I. Pupkin <vip@example.ru>
Date:   Sun Oct 18 11:57:55 2015 +0300
```

hello-world.txt: Update

```
commit d4ef969a680fae0286b47ced166abfd6b7df30c1
Author: Vasily I. Pupkin <vip@example.ru>
Date:   Sun Oct 18 05:52:22 2015 +0300
```

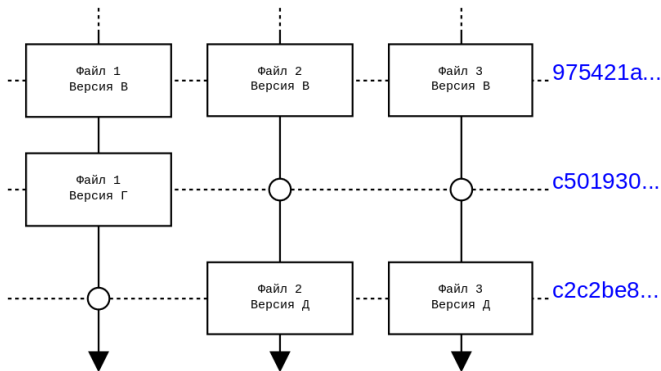
Initial commit

# “Родословная” коммитов



# Что представляют из себя коммиты?

## Слепки состояния (англ. *snapshots*)



И что?



Ну и зачем всё это?

# Ваша персональная машина времени!

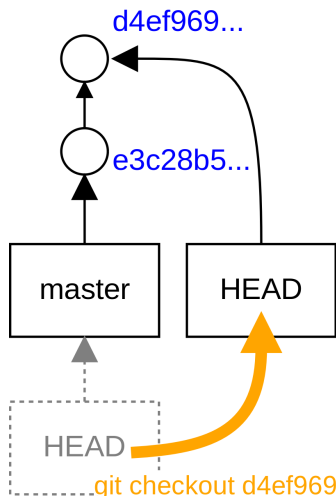
```
$ cat hello-world.txt  
hello  
world
```

**Переключение на коммит:**

```
$ git checkout d4ef969  
$ cat hello-world.txt  
hello
```

**Переключение на ветвь master:**

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



**Допустим, мы удалили важный файл:**

```
$ rm курсовая.tex
```

# Восстановление удалённых файлов





**Допустим, мы удалили важный файл:**

```
$ rm курсовая.tex
```

**Ой!..**

**Не беда, мы можем восстановить его из репозитория:**

```
$ git checkout курсовая.tex
```

## Переименование и перемещение файлов:

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

```
$ git status
```

На ветке master

Изменения, которые будут включены в коммит:

(используйте «git reset HEAD <файл>...»,  
чтобы убрать из индекса)

переименовано: hello-world.txt -> hello.txt

```
$ git commit -m "hello-world.txt: Rename"
```

## Удаление файлов:

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

```
$ git commit -m "hello-world.txt: Remove"
```

**Добавляем изменения в область подготовленных файлов:**

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

```
$ git status
```

На ветке master

Изменения, которые будут включены в коммит:

(используйте «git reset HEAD <файл>...»,  
чтобы убрать из индекса)

изменено:           hello-world.txt

**Что делать, если мы поторопились?**

Удаляем файл из области подготовленных файлов:

```
$ git reset HEAD hello-world.txt
```

```
$ git status
```

На ветке master

Изменения, которые не в индексе для коммита:

(используйте «git add <файл>...», чтобы добавить файл в индекс)

(используйте «git checkout -- <файл>...», чтобы отменить изменения в рабочем каталоге)

изменено:           hello-world.txt

нет изменений добавленных для коммита

(используйте «git add» и/или «git commit -a»)

# Отмена изменений в рабочем каталоге

**Осторожно! Здесь можно потерять незакоммиченные изменения!**

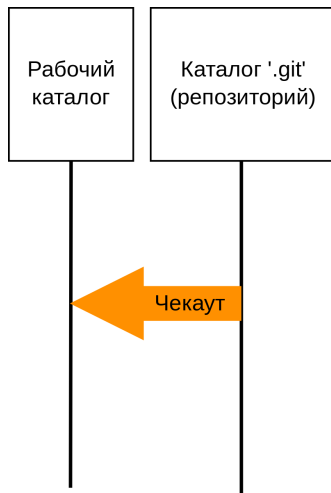
Что делать, если мы сделали изменения, которые нам больше не нужны и мы не хотим их коммитить? Отменить их!

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

```
$ git status
```

На ветке master

нечего коммитить, нет изменений  
в рабочем каталоге



## Клонируем репозиторий:

```
$ git clone \  
    https://github.com/artiom-poptsov/talks.git \  
    avp-talks
```

Клонирование в «avp-talks»...

remote: Counting objects: 161, done.

remote: Total 161 (delta 0), reused 0 (delta 0),  
 pack-reused 161

Получение объектов: 100% (161/161), 18.19 MiB | 2.73 MiB/s,  
 готово.

Определение изменений: 100% (48/48), готово.

Проверка соединения... готово.

## Переходим в каталог с проектом:

```
$ cd avp-talks
```

Подробнее про Git на русском языке:

- Scott Chacon, Ben Straub, **“Pro Git”** – <https://www.git-scm.com/book/ru/v1>
- Александр Швец, **“Git How To: Курс обучения Git на русском”** – <http://githowto.com/ru>
- Damir Shayhutdinov, **“Внутреннее устройство Git”** – [http://www.opennet.ru/base/dev/git\\_guts.txt.html](http://www.opennet.ru/base/dev/git_guts.txt.html)

Подробнее про системы управления версиями:

- Eric S. Raymond, **“Understanding Version-Control Systems”** – <http://www.catb.org/esr/writings/version-control/version-control.html>
- Rick Moen, **“Version-Control Systems for Linux”** – <http://linuxmafia.com/faq/Apps/vcs.html>

👉 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?



Фотографии котят и другие материалы с [wikimedia.org](https://commons.wikimedia.org/):

- Dwight Sipler, "You know, I think the large trees are easier. (411955683)" (CC-BY 2.0)
- That Guy, From That Show!, "A kitten opens its eyes for the first time" (PD)
- RN3DLL, "Scottish Kitten", (CC-BY 3.0)
- ColKorn1982, "Coll little Orange Tabby kitten" (CC-BY-SA 2.0)
- Dwight Sipler, "Packing for a trip" (CC-BY 2.0)
- Algerds, "Kitty meowing" (CC-BY-SA 3.0)
- Pagedelete, "Chizhik" (CC-BY-SA 3.0)
- Karin Dalziel, "Blue-eyed kitten" (CC-BY 2.0)
- Kerina yin, "Kucing belang perang (brown mackerel tabby cat)." (PD)
- Mysid, Incnis Mrsi, "Computer keyboard US" (PD)
- "Git logo" (CC-BY 3.0)

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/>