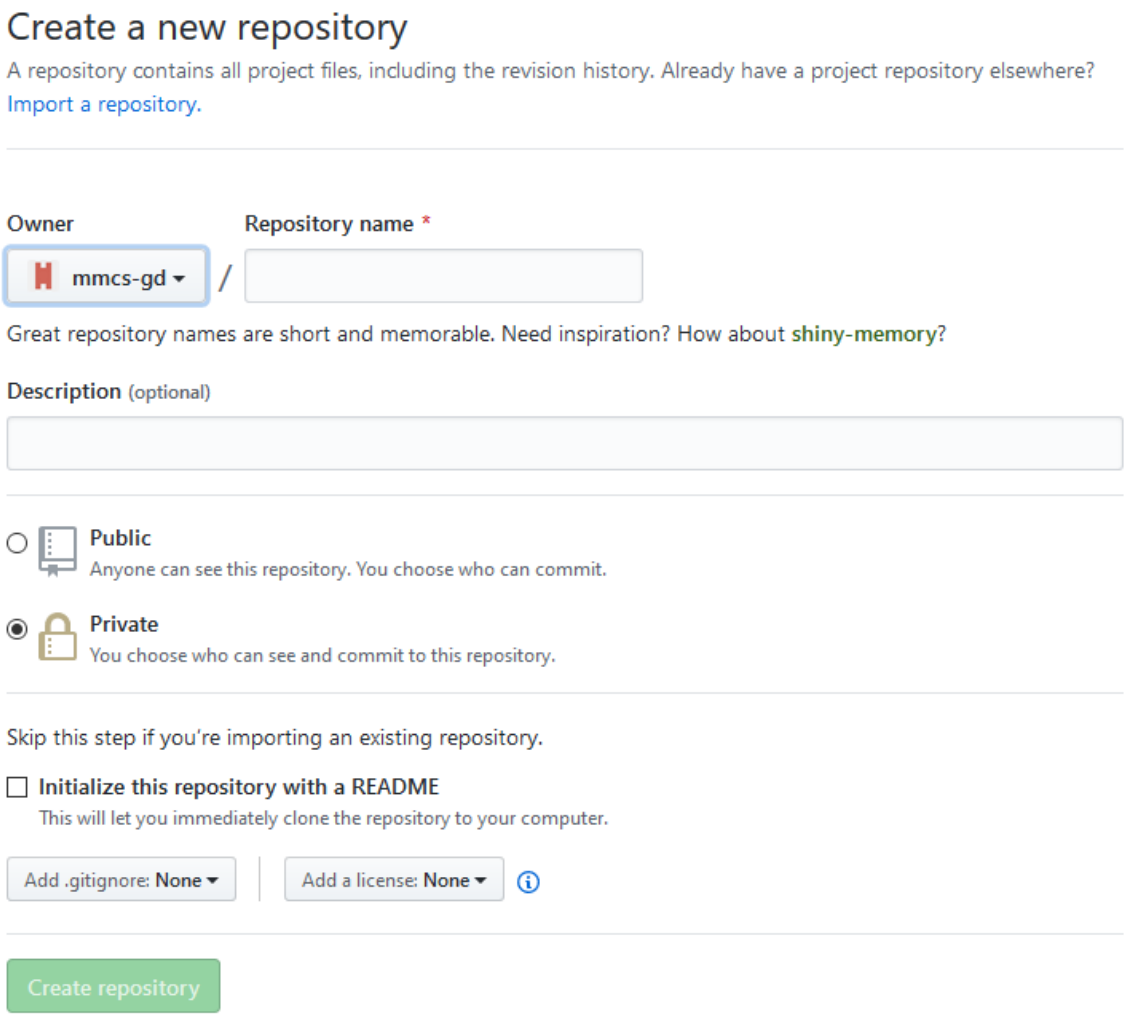
Git
Code Conventions
Лекция 4

План

- Git
- Blocks, lambdas, procs
- Code Conventions

С чего начать?



С чего начать?

$ git init

Создает каталог .git, в котором будет храниться информация о репозитории.

$ git clone

Получить удаленный репозиторий

Игнорируем лишнее: .gitignore

```
# комментарий – эта строка игнорируется
*.a
!lib.a
/TODO
# игнорировать все файлы в каталоге build/
build/

doc/*.txt

doc/**/*.txt
```

Как добавить файлы под контроль версий или в коммит?

```
$ git add .
$ git add *.rb
$ git add test.rb
```

Как сделать коммит?

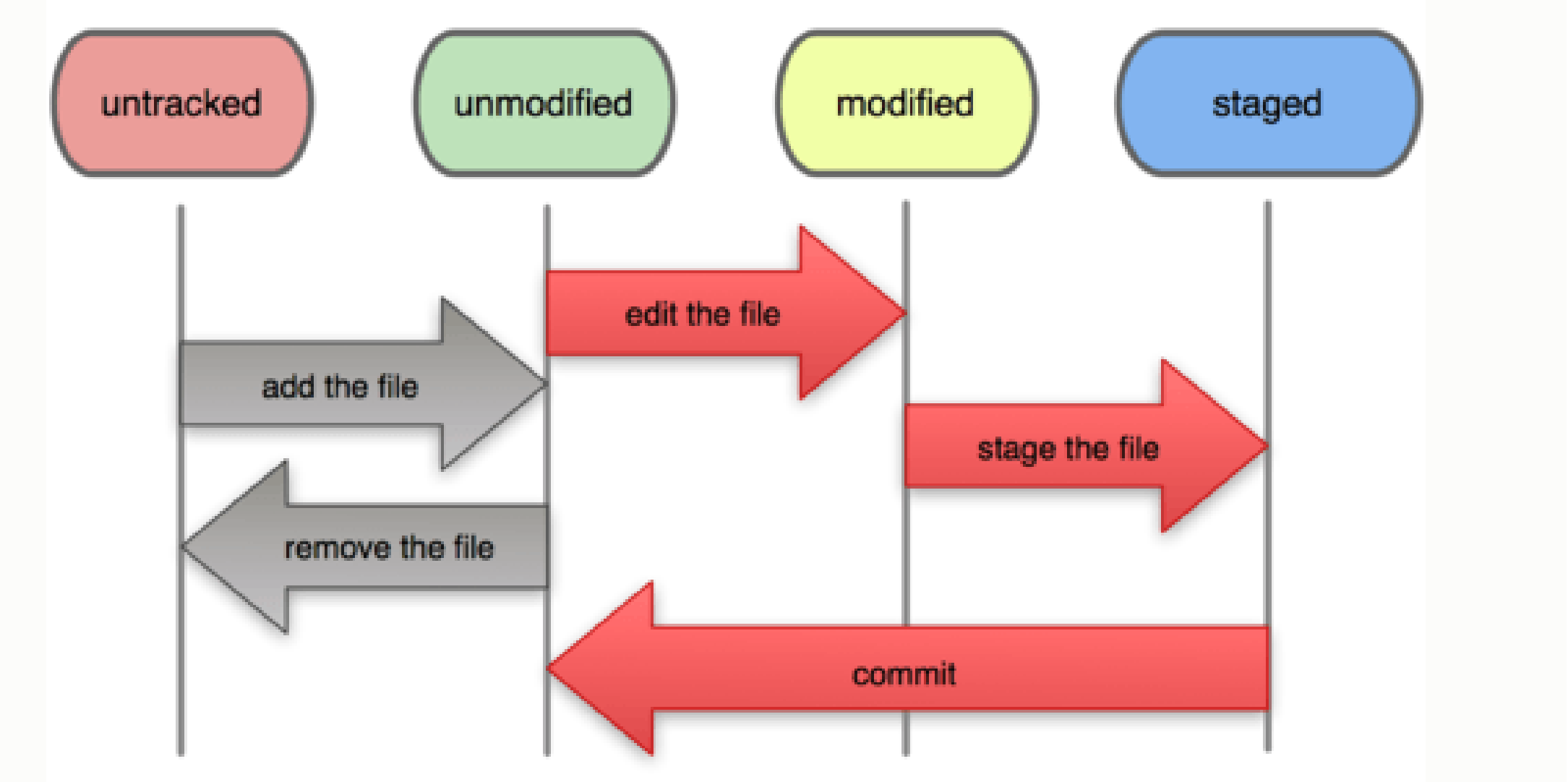
- `$ git commit -m "Initial commit"`

Как написать хорошее сообщение?

- Первая строка – короткое описание
- В императивном стиле
- Не более 72 символов
- Отвечает на вопросы «Что?» и «Почему?»
- Начинается с большой буквы
- Без точки в конце
- Если нужно описать подробности, добавьте пустую строку после первой строки, затем пишите

```
$ git commit –m “Creates project”
```

Жизненный цикл файла



Как связать локальный репозиторий с github?

```
$ git remote add
  origin https://github.com/user/repo.git
```

Как проверить состояние каталога?

$ git status

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

        new file:   app/assets/javascripts/test.js

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:   app/admin/card_templates.rb
        modified:   app/assets/javascripts/chart_control.js
        modified:   app/assets/javascripts/test.js
```

Как получить новую версию кода?

```
$ git pull
$ git pull -a

$ git fetch
$ git merge
```

Как отправить коммиты?

```
$ git push

$ git push my_branch

$ git push -u origin my_branch
```

Как создать новую ветку?

```
//в новую ветку
$ git checkout -b my_branch

//в существующую ветку
$ git checkout my_branch
```

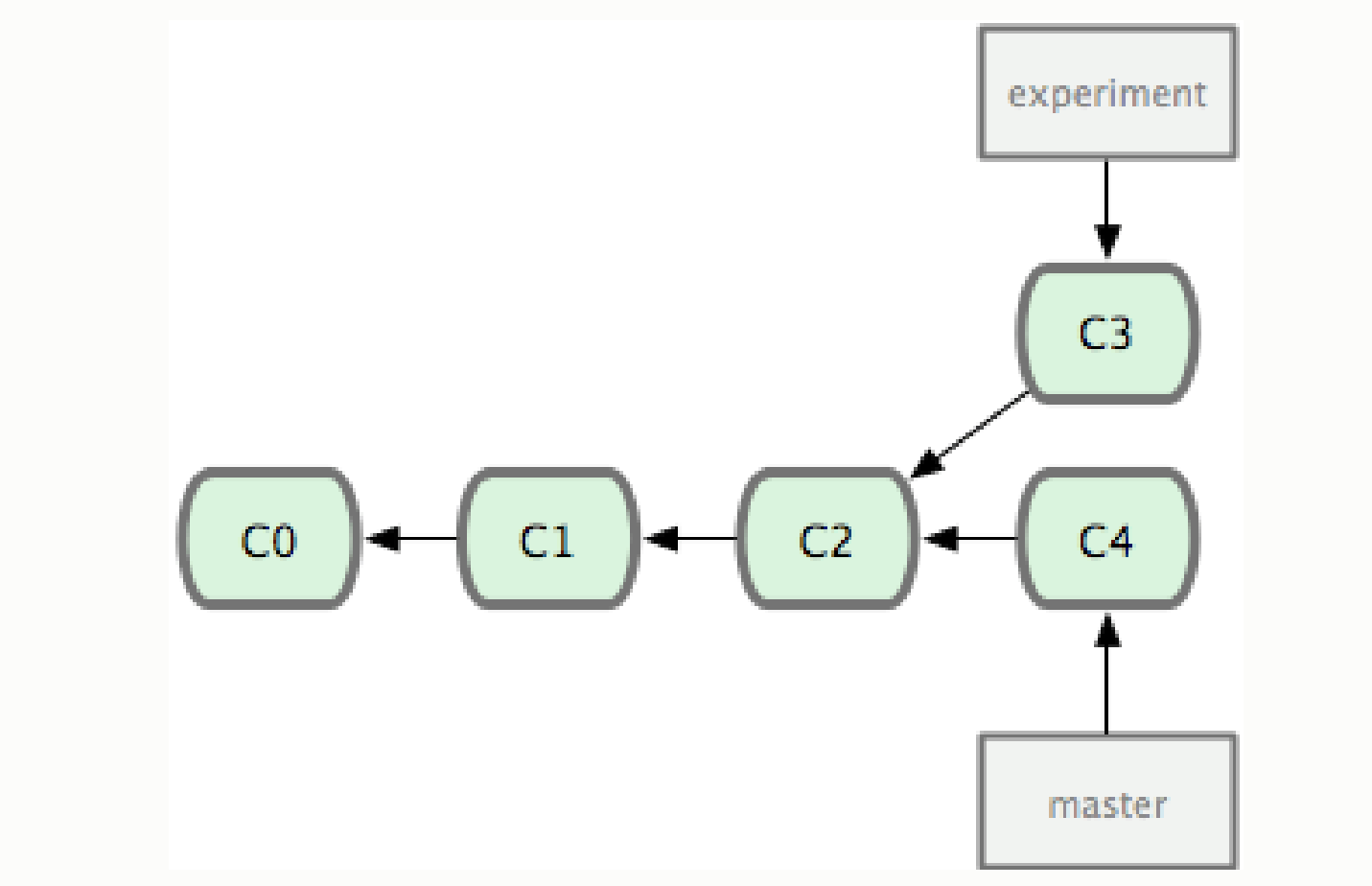
Рабочий процесс

```
$ git checkout master
$ git pull
$ git checkout -b task_31
$ git status
$ git add .
$ git commit -m “”
$ git push -u origin task_31
```

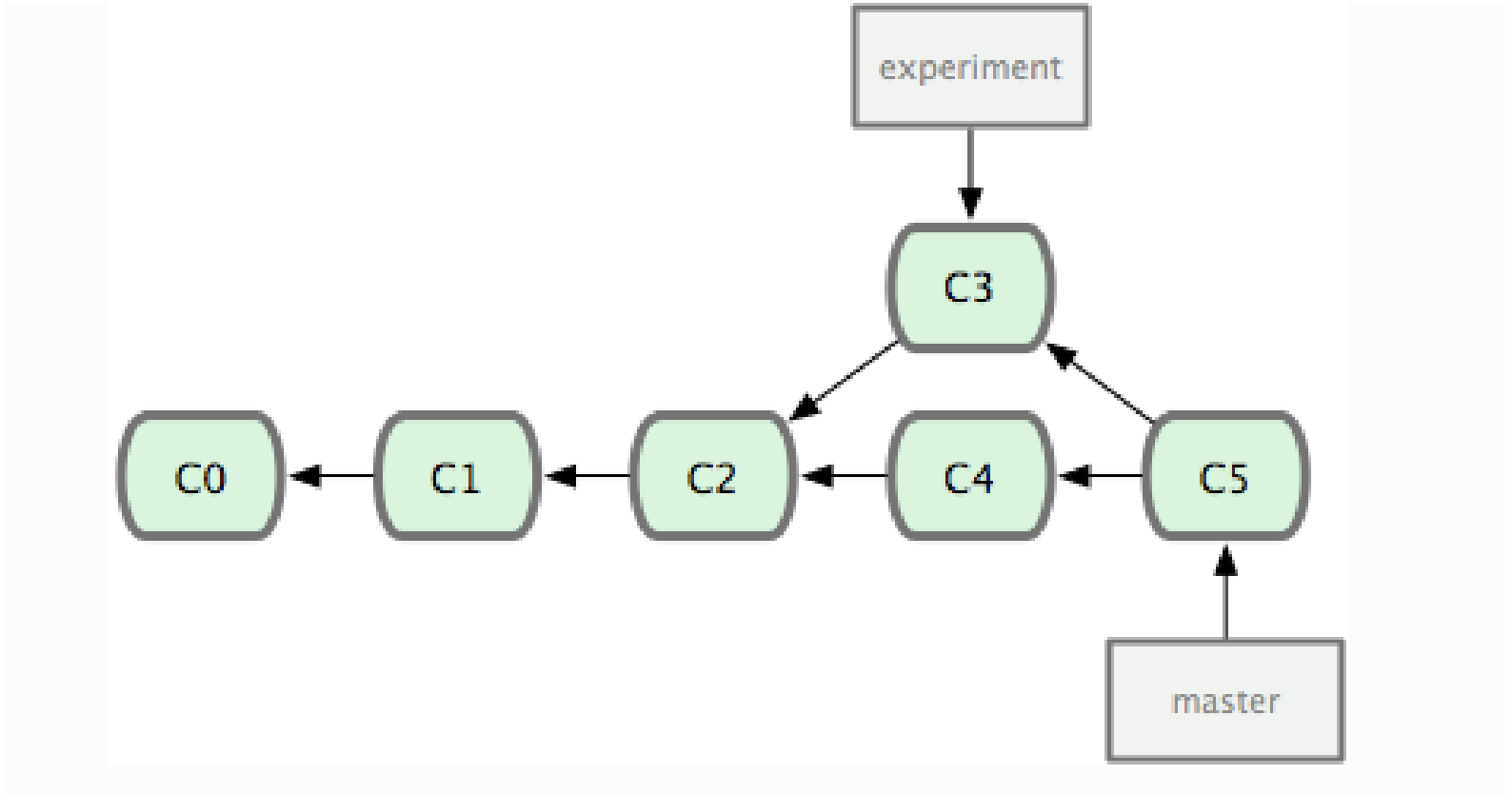
Рабочий процесс поумнее

```
...
$ git commit –m “”
$ git checkout master
$ git pull
$ git checkout task_31
$ git merge master
//разрешаем конфликты
```

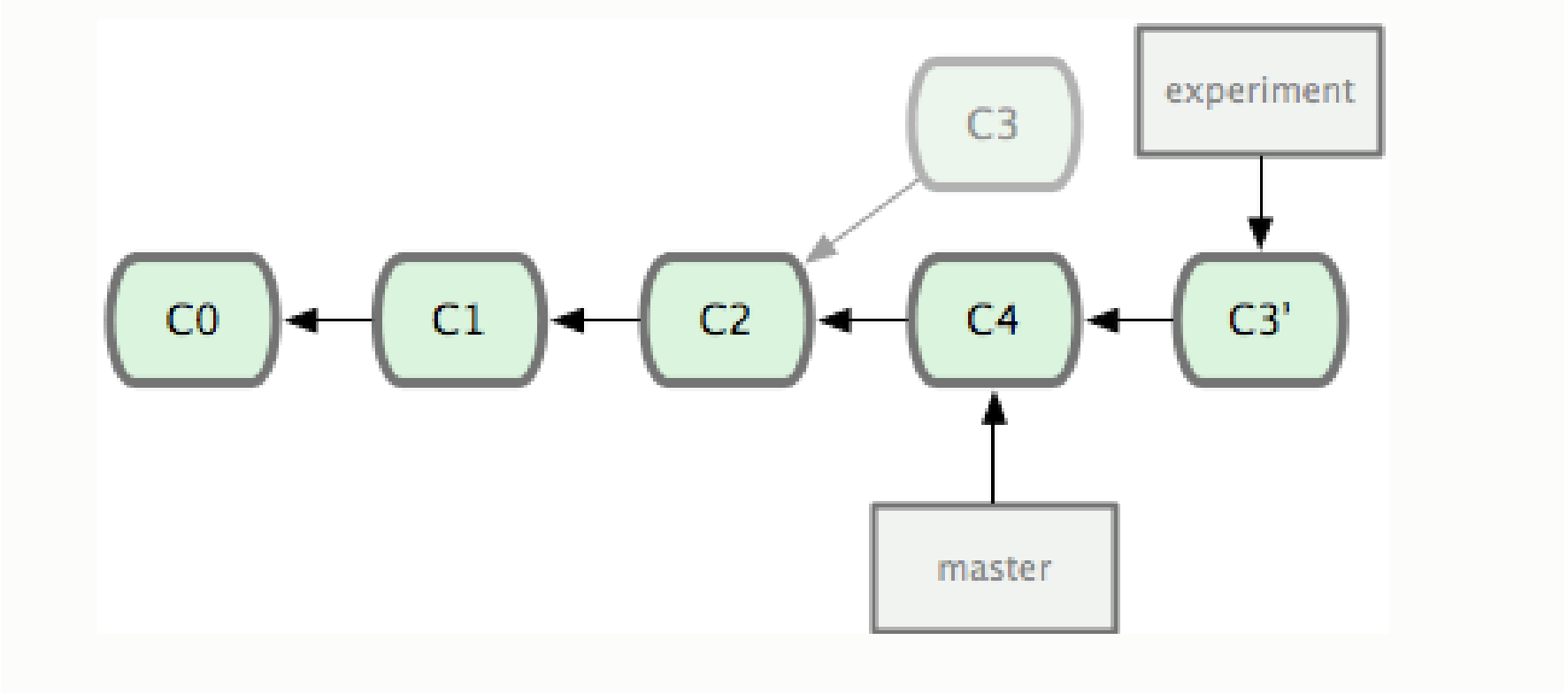
merge sv rebase



merge



rebase



История коммитов

```
$ git log
$ git log --stat
$ git log --pretty=format:"%h - %an, %ar : %s"
$ git log --graph
$ git log –since=2.weeks
```

Pretty?

%H	Хеш коммита
%h	Сокращённый хеш коммита
%T	Хеш дерева
%t	Сокращённый хеш дерева
%P	Хеши родительских коммитов
%p	Сокращённые хеши родительских коммитов
%an	Имя автора
%ae	Электронная почта автора
%ad	Дата автора (формат соответствует параметру --date=)
%ar	Дата автора, относительная (пр. "2 мес. назад")
%cn	Имя коммитера
%ce	Электронная почта коммитера
%cd	Дата коммита
%cr	Дата коммита, относительная
%s	Комментарий

Если что-то пошло не так

Убрать подготовленный файл

```
$ git reset HEAD test.rb
```

Переименовать коммит

```
$ git commit -amend
```

Поправить историю коммитов

```
$ git rebase -i HEAD~5
```

Что можно сделать с коммитом?

```
# p, pick = use commit
# r, reword = use commit,
but edit the commit message
# e, edit = use commit, but
stop for amending
```

Что можно сделать с коммитом?

```
# 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
# d, drop = remove commit
```

Ищем проблему

```
$ git blame test.rb
```

Еще раз про процесс

- Клонируем нашу копию
- Создаём тематическую ветку
- Вносим свои изменения
- Проверяем изменения
- Фиксируем изменения в тематической ветку
- Отправляем новую ветку в нашу копию на GitHub

Блоки, проки и лямбды

Blocks, procs, lambda

Блоки

```ruby
[1, 2, 3].each { |num| puts num }
```

```ruby
[1, 2, 3].each do |num|
  puts num
  a-=num
end
```

```ruby
def print_once
  yield
end
print_once { puts "Block is being run" }
```

yeild с параметром

```ruby
def one_two_three
  yield 1
  yield 2
  yield 3
end
one_two_three { |number| puts number * 10 }
# 10, 20, 30
```

Явные блоки

```ruby
def explicit_block(&block)
  block.call # same as yield
end
explicit_block { puts "Explicit block called" }
```

Что если блока нет?

```ruby
def do_something_with_block
  return "No block given" unless block_given?
  yield
end
```

Lambda

```ruby
say_something = -> { puts "This is a lambda" }
# or
say_something = lambda { puts "This is a lambda" }
say_something.call
```

```ruby
my_lambda.()
my_lambda[]
my_lambda.===
```

Аргументы

```
times_two = ->(x) { x * 2 }
times_two.call(10)
# 20
```

Proc

```ruby
my_proc = Proc.new { |x| puts x }

t = Proc.new { |x,y| puts "I don't care about arguments!" }
t.call
```

Выход из Lambda и Proc

```ruby
my_lambda = -> { return 1 }
puts "Lambda result: #{my_lambda.call}"

my_proc = Proc.new { return 1 }
puts "Proc result: #{my_proc.call}"
```

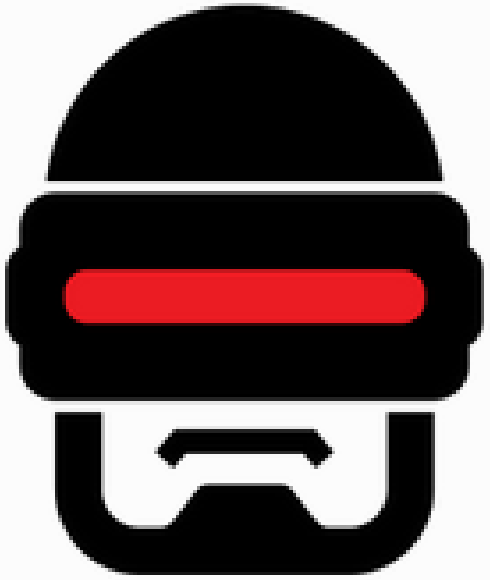
Замыкания

- Захват контекста
- Ссылки на переменные

```ruby
def call_proc(my_proc)
  count = 500
  my_proc.call
end

count   = 1
my_proc = Proc.new { puts count }
call_proc(my_proc)
```

Я тоже хочу замыкать!

```ruby
def return_binding
  foo = 100
  binding #!!!
end

puts return_binding.class
puts return_binding.eval('foo')

puts foo
```



RuboCop

Role models are important.
Alex J. Murphy

Зачем styleguide?

- Целостность важна
- Целостность внутри гайда важна
- Целостность внутри проекта важнее
- Целостность внутри класса еще важнее

Когда нарушать?

- Поддержка старых версий Ruby
- Читаемость
- Окружающий код

Оформление кода

Табы или пробелы?
Размер отступа

```ruby
# bad - four spaces
def some_method
    do_something
end

# good - 2 spaces
def some_method
  do_something
end
```

Оформление строки кода

- До 80 символов
- Без пробелов на конце строки
- Unix-like концы строк

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

Конец файла

- Пустая строка в конце каждого файла

```ruby
;

# bad
puts 'foobar'; # superfluous semicolon

# good
puts 'foobar'
```

Одно выражение на строке

```ruby
# bad
puts 'foo'; puts 'bar' # two expressions on the same line

# good
puts 'foo'
puts 'bar'

puts 'foo', 'bar' # this applies to puts in particular
```

Пробелы вокруг операторов

```
sum = 1 + 2
a, b = 1, 2
class FooError < StandardError; end
```

Пробелы вокруг операторов

```ruby
# bad
e = M * c ** 2

# good
e = M * c**2
```

```ruby
# bad
o_scale = 1 / 48r

# good
o_scale = 1/48r
```

Safe navigation operator

```
# bad
foo &. bar
foo &.bar
foo&. bar

# good
foo&.bar
```

Safe navigation operator ???

```ruby
class Blazon
  def show
    puts "\u2694".encode('utf-8')
  end
end

class Shield
  attr_accessor :blazon

  def initialize(blazon = nil)
    @blazon = blazon
  end
end
```

```ruby
class Warrior

  attr_accessor :shield

  def initialize(shield = nil)
    @shield = shield
  end
end
```

Safe navigation operator ???

```
warrior = Warrior.new(Shield.new(Blazon.new))
warrior.shield.blazon.show
```

```
warrior_without_shield = Warrior.new
warrior_with_empty_shield = Warrior.new(Shield.new)
```

```
Uncaught exception: undefined method `blazon' for
nil:NilClass
        C:/Code/ruby_tests/test.rb:29:in `<main>'
```

Safe navigation operator ???

```ruby
if warrior && warrior.shield && warrior.shield.blazon
  warrior.shield.blazon.show
end
```

```ruby
warrior&.shield&.blazon&.show
```

Скобки и пробелы

```ruby
# bad
some( arg ).other
[ 1, 2, 3 ].each{|e| puts e}

# good
some(arg).other
[1, 2, 3].each { |e| puts e }
```

{ } многозначны

```
# good - space after { and before }
{ one: 1, two: 2 }

# good - no space after { and before }
{one: 1, two: 2}
```

{ } многозначны

```ruby
# bad
"From: #{ user.first_name }, #{ user.last_name }"

# good
"From: #{user.first_name}, #{user.last_name}"
```

Без пробелов

```
# bad
! something

# good
!something
```

```
# bad
1 .. 3
'a' ... 'z'

# good
1..3
'a'...'z'
```

Отступы в case

```ruby
# good
case
when song.name == 'Misty'
  puts 'Not again!'
when song.duration > 120
  puts 'Too long!'
when Time.now.hour > 21
  puts "It's too late"
else
  song.play
end
```

Присваивание условных операторов

```ruby
kind = case year
       when 1850..1889 then 'Blues'
       when 1890..1909 then 'Ragtime'
       when 1910..1929 then 'New Orleans Jazz'
       when 1930..1939 then 'Swing'
       when 1940..1950 then 'Bebop'
       else 'Jazz'
       end

result = if some_cond
           calc_something
         else
           calc_something_else
         end
```

Пустые строки

- После методов
- Вокруг модификаторов доступа
- Не более 1 пустой строки
- Не нужны вокруг тел классов, методов, модулей

Запятая в конце списка параметров

```
# bad
some_method(size, count, color, )

# good
some_method(size, count, color)
```

Склеивание строк кода

```
# bad
result = 1 - \
         2

# good (but still ugly as hell)
result = 1 \
         - 2

long_string = 'First part of the long string' \
              ' and second part of the long string'
```

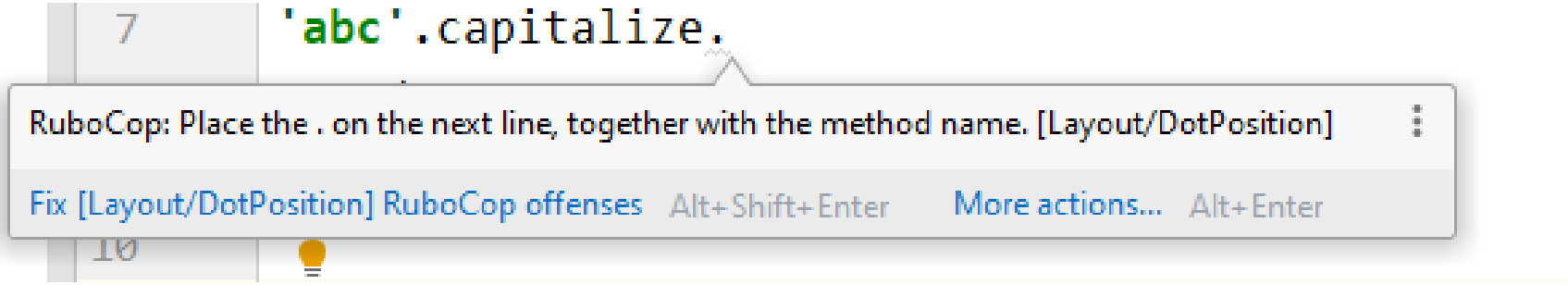
Method Chaining

```ruby
#bad
one.two.three.
    four
```

```ruby
# bad
one.two.three
    .four
```

```ruby
# good
one.two.three
    .four
```

```ruby
# good
one.two.three.
    four
```



Соглашение о наименовании

snake_case

```ruby
# good
:some_symbol

some_var = 5

def some_method
  # some code
end
```

```ruby
# good
:some_sym1

some_var1 = 1

var10 = 10

def some_method1
  # some code
end
```

- Имена файлов
- Имена папок
- Кричащие константы MY_CONST

? и !

- ? в конце булевских методов
- Без служебных глаголов (`empty?`, а не `is_empty?`)
- ! – меняет вызывающий объект или аргументы
- Сделайте и безопасный метод тоже

Пример безопасного метода

```ruby
class Array
  def flatten_once!
    res = []

    each do |e|
      [*e].each { |f| res << f }
    end

    replace(res)
  end
end
```

```ruby
def flatten_once
  dup.flatten_once!
end
```

```ruby
a = [1,2,[3,4]]
a.flatten_once!
puts a
```

Имена классов

```ruby
# good
class SomeClass
  # some code
end

class SomeXML
  # some code
end

class XMLSomething
  # some code
end
```

Подчеркивание для ненужных переменных

```ruby
# good
result = hash.map { |_, v| v + 1 }

def something(x)
  _, used_var = something_else(x)
  # some code
end
```

Other

- Для симметричных бинарных операторов второй операнд должен называться other

```ruby
# good
def +(other)
  # body omitted
end

# bad
def <<(other)
  @internal << other
end
```

Поток управления

Не используйте for

- for в Ruby не создает блок!

```ruby
arr = [1, 2, 3]

# bad
for elem in arr do
  puts elem
end

elem # => 3
```

```ruby
# good
arr.each { |elem| puts elem }

elem # => NameError: undefined
local variable or method `elem
```

Then не нужен

```ruby
# good
if some_condition
  # body omitted
end

# good
case foo
when bar
  # body omitted
end
```

Тернарный оператор

- Лучше, чем if
- Только один уровень вложенности
- Только 1 строка

```
# bad
some_condition ? (nested_condition ? nested_something : nested_som

# good
if some_condition
  nested_condition ? nested_something : nested_something_else
else
  something_else
end
```

if, unless, ||, &&, !

- ! лучше not
- || и && лучше or и and
- Однострочные if/unless
- !! не нужно
- unless лучше !

while, unless, loop

```ruby
# good
do_something while some_condition
do_something until some_condition

loop do
  break unless val < 0
end
```

return

- Последнее выражение возвращается
- Явно писать не нужно
- guard clause нужны

Nested condition

```ruby
# bad
def compute_thing(thing)
  if thing[:foo]
    update_with_bar(thing[:foo])
    if thing[:foo][:bar]
      partial_compute(thing)
    else
      re_compute(thing)
    end
  end
end
```

Guard Clause

```ruby
# good
def compute_thing(thing)
  return unless thing[:foo]
  update_with_bar(thing[:foo])
  return re_compute(thing) unless thing[:foo][:bar]
  partial_compute(thing)
end
```

Исключения

Не используйте return в ensure

```ruby
# bad
def foo
  raise
ensure
  return 'very bad idea'
end
```

Неявный begin

```ruby
# good
def foo
  # main logic goes here
rescue
  # failure handling goes here
end
```

Методы непредвиденных обстоятельств

```ruby
# bad
begin
  something_that_might_fail
rescue IOError
  # handle IOError
end

begin
  something_else_that_might_fail
rescue IOError
  # handle IOError
end
```

Методы непредвиденных обстоятельств

```ruby
# good
def with_io_error_handling
  yield
rescue IOError
  # handle IOError
end

with_io_error_handling { something_that_might_fail }

with_io_error_handling { something_else_that_might_fail }
```

Общие замечания

- Не прячьте исключения
- Исключения – это не flow control
- Явно указывайте класс исключения в `rescue`
- Не ловите `Exception`
- В `ensure` можно освободить ресурсы

Сравнение и присваивание

Параллельное присваивание

```
# bad
a, b, c, d = 'foo', 'bar', 'baz', 'foobar'

# good
a = 'foo'
b = 'bar'
c = 'baz'
d = 'foobar'
```

Параллельное присваивание

- Для swap, для возвращаемых значений, для splat

```ruby
a = 'foo'
b = 'bar'

a, b = b, a
puts a # => 'bar'
puts b # => 'foo'

# good - method return
def multi_return
  [1, 2]
end

first, second = multi_return
```

```ruby
# good - use with splat
first, *list = [1, 2, 3, 4] # first => 1, list
=> [2, 3, 4]

hello_array = *'Hello' # => ["Hello"]

a = *(1..3) # => [1, 2, 3]
```

Сокращения для самоприсваивания

```ruby
# good
x += y
x *= y
x **= y
x /= y
x ||= y
x &&= y
```

```ruby
name ||= 'wow'

something &&= something.downcase
```

BLP StyleGuide

```ruby
names.map(&:upcase)
names.select { |name| name.start_with?('S') }.map(&:upcase)
```

```ruby
# bad
l = lambda { |a, b| a + b }
l.call(1, 2)

l = ->(a, b) do
  tmp = a * 7
  tmp * b / 50
end
```

```ruby
# good
l = ->(a, b) { a + b }
l.call(1, 2)

l = lambda do |a, b|
  tmp = a * 7
  tmp * b / 50
end
```

BLP StyleGuide

```ruby
l = ->(x, y) { something(x, y) }
l = -> { something }
```

```ruby
# bad
p = Proc.new { |n| puts n }

# good
p = proc { |n| puts n }
```

Методы

Общие положения

- Не более 10 строк
- **def** и **end** на разных строках
- **::** для констант, **.** для методов
- **()** круглые скобки для списка параметров
- () для вызова с параметрами
- Не более 4 параметров
- Параметры по умолчанию в конце списка параметров

DSL, Keyword, super

```
validates :name, presence: true

attr_reader :name, :age

puts person.age

super(name, age)
```

Именованные параметры

```ruby
# good
def some_method(bar: false)
  puts bar
end

some_method                 # => false
some_method(bar: true)
```

Именованные параметры

```ruby
# bad
def some_method(a, b = 5, c = 1)
  # body omitted
end

# good
def some_method(a, b: 5, c: 1)
  # body omitted
end
```

Именованные параметры

```ruby
# bad
def some_method(options = {})
  bar = options.fetch(:bar, false)
  puts bar
end

# good
def some_method(bar: false)
  puts bar
end
```

Метод с глобальной областью видимости

- Не нужны
- Если нужны, то в Kernel.

Ссылки

- https://education.github.com/git-cheat-sheet-education.pdf
- https://git-scm.com
- https://github.com/rubocop-hq/ruby-style-guide
- https://www.rubyguides.com/2016/02/ruby-procs-and-lambdas/