

Системы управления версиями. GIT

Наумов Д.А., доц. каф. КТ

Содержание лекции

- 1 Управление версиями
- 2 Операции в GIT
- 3 Основы работы в GIT
- 4 Работа с удаленными репозиториями
- 5 Ветвление в Git
- 6 Основы ветвления и слияния. Пример
- 7 Основы разрешения конфликтов слияния

Введение

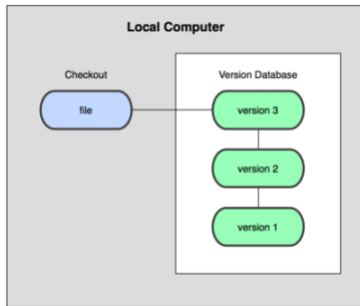
Система управления версиями

система, сохраняющая изменения в одном или нескольких файлах так, чтобы потом можно было восстановить определённые старые версии.

Задачи, решаемые СУВ:

- вернуть файлы к прежнему виду;
- вернуть к прежнему состоянию весь проект;
- сравнить изменения с какого-то времени;
- увидеть, кто последним изменял модуль, который дал сбой, кто создал проблему;
- восстановить проект в любом состоянии.

Система управления версиями



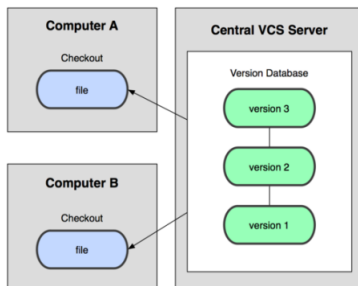
Особенность локальных СУВ:

- имеется база данных, в которой хранятся все изменения нужных файлов;
- базы данных хранится на локальном компьютере;
- изменения хранятся в виде набора патчей между парой изменений;

Пример: утилита *rcs*.

Цель: ЦСУВ - сотрудничество удаленных разработчиков.

Централизованная система управления версиями



Особенность ЦСУВ:

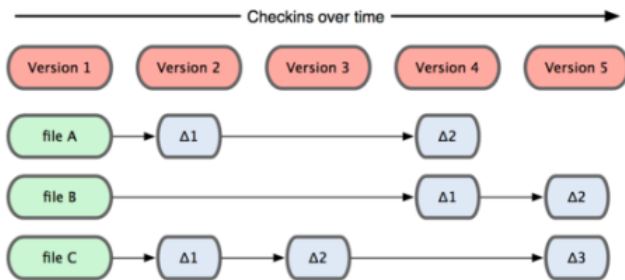
- все отслеживаемые файлы хранятся на сервере;
- клиенты получают копии файлов с сервера;

Пример: *CSV*, *Subversion*, *Perforce*.

- + контроль за разработчиками; проще администрирование;
- - взаимодействие без сервера невозможно; потеря истории при повреждении центральной БД.

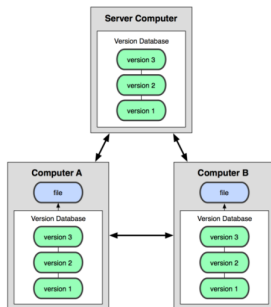
Хранение изменений файлов в ЦСУВ

Данные хранятся как изменения к базовой версии для каждого файла



Хранимые данные - набор файлов и изменений, сделанных для каждого из этих файлов во времени.

Распределенная система управления версиями



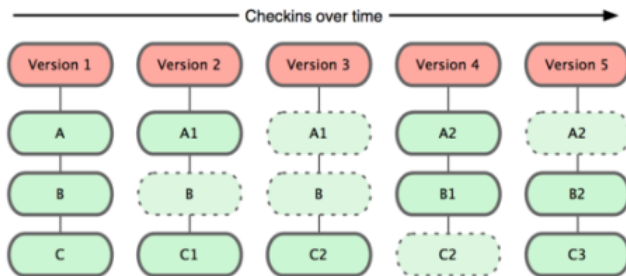
Особенность РСУВ:

- клиенты полностью копируют репозиторий;
- клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить БД;
- есть возможность работать с несколькими удаленными репозиториями;

Пример: *Git*, *Mercurial*, *Bazaar*, *Darcs*.

Хранение изменений файлов в РСУВ

Данные хранятся как слепки состояния проекта



Хранимые данные - набор слепков состояния файловой системы.

Операции в GIT

Для большинства операций не нужно соединение с сетью:

- просмотр истории проекта;
- просмотр различий версий файла (например, сейчас и месяц назад);
- возможность сохранения изменений в БД

Контроль целостности данных:

- вычисление контрольных сумм SHA-1;
- невозможно "незаметно" изменить файл или каталог;

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Данные в системы только добавляются. Сложно заставить GIT удалить данные или сделать что-то неотменяемое.

Состояние файлов в GIT

В Git файлы могут находиться в одном из трёх состояний:

- зафиксированном - файлы уже сохранены в локальной базе;
- изменённом - файлы, которые поменялись, но ещё не были зафиксированы;
- подготовленном - изменённые файлы, отмеченные для включения в следующий коммит.

Стандартный рабочий процесс:

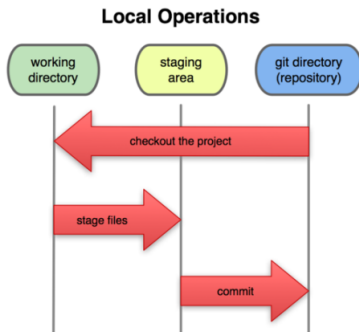
1. Вы изменяете файлы в вашем рабочем каталоге.

2. Вы подготавливаете файлы, добавляя их слепки в область подготовленных файлов.

3. Вы делаете коммит. При этом слепки из области подготовленных файлов сохраняются в каталог Git.

Три части проекта с использованием GIT:

- каталог Git — место, где Git хранит метаданные и базу данных объектов проекта;
- рабочий каталог — это извлечённая из базы копия определённой версии проекта;
- область подготовленных файлов — файл, хранящийся в каталоге Git, содержащий информацию о том, что должно войти в следующий коммит.



Установка GIT

- в Linux (Debian): *apt-get install git-core*
- в Windows: *<http://code.google.com/p/msysgit>*
- в Mac: *<http://code.google.com/p/git-osx-installer>*

Настройка GIT: git config

- файл `/etc/gitconfig` содержит значения, общие для всех пользователей вашей системы и всех их репозиториях (`-system`);
- Файл `/.gitconfig` хранит настройки конкретного пользователя (`-global`);
- конфигурационный файл в каталоге Git (`.git/config`) в том репозитории, где вы находитесь в данный момент

Настройка пользователя

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

Настройка редактора

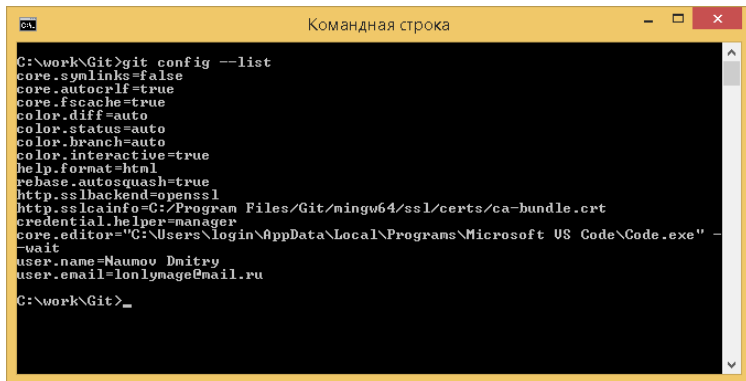
```
$ git config --global core.editor emacs
```

Настройка утилиты сравнения

```
$ git config --global merge.tool vimdiff
```

Просмотр настроек

```
$ git config --list
```



```
C:\work\Git>git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
rebase.autosquash=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
credential.helper=manager
core.editor="C:\Users\login\AppData\Local\Programs\Microsoft US Code\Code.exe" -
-wait
user.name=Naumov Dmitry
user.email=lonlymage@mail.ru

C:\work\Git>_
```

Получение помощи

```
$ git help <команда>  
$ git <команда> --help  
$ man git-<команда>
```

Получение помощи по конкретной команде

```
$ git help config
```

Создание репозитория Git

Новый репозиторий в текущем каталоге

```
$ git init
```

```
$ git add *.c  
$ git add README  
$ git commit -m 'initial project version'
```

Клонирование существующего репозитория

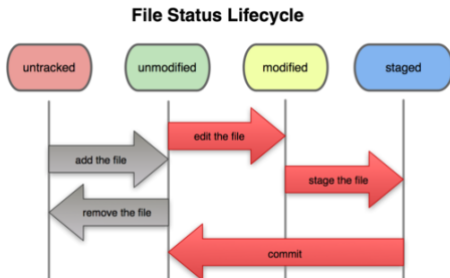
```
$ git clone git://github.com/schacon/grit.git mygrit
```


Запись изменений в репозиторий

Файлы в рабочем каталоге могут быть в состоянии:

- отслеживаемые - под версионным контролем (неизмененные, измененные, подготовленными к коммиту).
- неотслеживаемые - любые файлы, которые не входили в последний слепок состояния и не подготовлены к коммиту.

Жизненный цикл состояния файлов проекта



Определение состояний файлов

git status

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

git status после создания нового файла

```
$ vim README
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
# README
nothing added to commit but untracked files present (use "git add" to track)
```

Отслеживание новых файлов

git add

```
$ git add README
```

git status после добавления отслеживания

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

Индексация отслеживаемых файлов

изменим `benchmarks.rb`, потом - `git status`

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# new file:   README
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#
# modified:   benchmarks.rb
#
```

«Changed but not updated» — отслеживаемый файл был изменен в рабочем каталоге, но пока не проиндексирован.

Индексация отслеживаемых файлов

git add, потом - git status

```
$ git add benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# new file:   README
# modified:   benchmarks.rb
#
```

Оба файла проиндексированы и войдут в следующий коммит.

Индексация отслеживаемых файлов

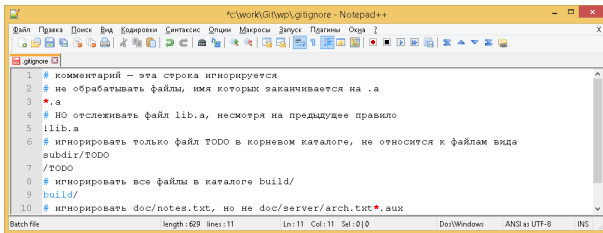
снова изменим файл и проверим - git status

```
$ vim benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   README
#   modified:   benchmarks.rb
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#
#   modified:   benchmarks.rb
#
```

benchmarks.rb - проиндексированный и неиндексированный одновременно.

Игнорирование файлов

Файл .gitignore



```
*c:\work\Git\wp\.gitignore - Notepad++
1 # комментарий — эта строка игнорируется
2 # не обрабатывать файлы, имя которых заканчивается на .a
3 *.a
4 # но отслеживать файл lib.a, несмотря на предыдущее правило
5 !lib.a
6 # игнорировать только файл TODO в корневом каталоге, не относится к файлам вида
  subdir/TOD0
7 /TOD0
8 # игнорировать все файлы в каталоге build/
9 build/
10 # игнорировать doc/notes.txt, но не doc/server/arch.txt*.aux
```

Glob-шаблоны (упрощенные регулярные выражения:

- * - соответствует 0 или более символам;
- [abc] — соответствует любому символу из указанных в скобках (в данном случае a,b,c);
- ? - соответствует одному символу;
- [0 — 9] - соответствует любому символу из интервала (в данном случае от 0 до 9).

Просмотр различий в файлах

Команда *git diff* сравнивает содержимое вашего рабочего каталога с содержимым индекса. Результат показывает еще не проиндексированные изменения.

git diff

```
$ git diff
diff --git a/benchmarks.rb b/benchmarks.rb
index 3cb747f..da65585 100644
--- a/benchmarks.rb
+++ b/benchmarks.rb
@@ -36,6 +36,10 @@ def main
     @commit.parents[0].parents[0].parents[0]
   end

+  run_code(x, 'commits 1') do
+    git.commits.size
+  end
+
   run_code(x, 'commits 2') do
     log = git.commits('master', 15)
     log.size
```


Фиксация изменений

git commit

```
$ git commit
```

Эта команда откроет выбранный текстовый редактор:

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#       modified:   benchmarks.rb
~
~
~
".git/COMMIT_EDITMSG" 10L, 283C
```

Фиксация изменений

git commit -m текст сообщения

```
$ git commit -m "Story 182: Fix benchmarks for speed"
[master]: created 463dc4f: "Fix benchmarks for speed"
2 files changed, 3 insertions(+), 0 deletions(-)
create mode 100644 README
```

Коммит вывел информацию:

- на какую ветку вы выполнили коммит (master);
- какая контрольная сумма SHA-1 у этого коммита (463dc4f);
- сколько файлов было изменено;
- статистику по добавленным/удаленным строкам в этом коммите.

Игнорирование индексации

`git commit -a -m текст сообщения`

```
$ git status
# On branch master
#
# Changed but not updated:
#
#   modified:   benchmarks.rb
#
$ git commit -a -m 'added new benchmarks'
[master 83e38c7] added new benchmarks
1 files changed, 5 insertions(+), 0 deletions(-)
```

Удаление файла

Для того чтобы удалить файл из Git, необходимо удалить его из отслеживаемых файлов (точнее, удалить его из индекса), а затем выполнить КОММИТ.

`git rm имяфайла`

```
$ git rm grit.gemspec
rm 'grit.gemspec'
$ git status
# On branch master
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       deleted:    grit.gemspec
#
```

Если просто удалить файл из рабочего каталога, он будет показан в секции «Changed but not updated».

Удаление файла

git rm --cached имяфайла

```
$ git rm --cached readme.txt
```

использование шаблонов

```
$ git rm log/*.log
```

```
$ git rm \*~
```

Перемещение файла

git mv

```
$ git mv file_from file_to
```

результат перемещения

```
$ git mv README.txt README
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       renamed:    README.txt -> README
#
```

Просмотр истории коммитов

git log

```
git clone git://github.com/schacon/simplegit-progit.git
```

результат выполнения команды

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700
```

changed the version number

```
commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700
```

removed unnecessary test code

```
commit a11be506235650402fa7552ab500cd00dc23000e6
```

Отмена последнего коммита

git commit --amend

```
$ git commit --amend
```

второй коммит заменяет результаты первого

```
$ git commit -m 'initial commit'  
$ git add forgotten_file  
$ git commit --amend
```


Отмена индексации файла

git reset HEAD file

```
$ git reset HEAD benchmarks.rb
benchmarks.rb: locally modified
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README.txt
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   benchmarks.rb
#
```

Отмена изменения файла: git checkout – benchmarks.rb

Работа с удаленными репозиториями

Удаленный репозиторий

модификации проекта, которые хранятся в интернете или ещё где-то в сети.

Совместная работа включает в себя управление удалёнными репозиториями и помещение (push) и получение (pull) данных в и из них тогда, когда нужно обменяться результатами работы. Управление удалёнными репозиториями включает:

- умение добавлять удалённые репозитории;
- умение удалять те из них, которые больше не действуют;
- умение управлять различными удалёнными ветками;
- умение определять их отслеживание.

Отображение удаленных репозиторияев

git remote

```
$ git clone git://github.com/schacon/ticgit.git
```

```
$ cd grit
$ git remote -v
bakkdoor  git://github.com/bakkdoor/grit.git
cho45     git://github.com/cho45/grit.git
defunkt   git://github.com/defunkt/grit.git
koke      git://github.com/koke/grit.git
origin    git@github.com:mojombo/grit.git
```

Добавление удаленных репозиторий

Добавить новый удалённый Git-репозиторий под именем-сокращением *git remote add*:

```
$ git remote
origin
$ git remote add pb git://github.com/paulboone/ticgit.git
$ git remote -v
origin git://github.com/schacon/ticgit.git
pb git://github.com/paulboone/ticgit.git
```

Теперь вы можете использовать в командной строке имя `pb` вместо полного URL.

```
$ git fetch pb
remote: Counting objects: 58, done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 44 (delta 24), reused 1 (delta 0)
Unpacking objects: 100% (44/44), done.
From git://github.com/paulboone/ticgit
* [new branch]      master      -> pb/master
* [new branch]      ticgit      -> pb/ticgit
```

fetch

Получение данных из удаленного репозитория:

```
$ git fetch [remote-name]
```

Данная команда:

- связывается с указанным удалённым проектом;
- забирает все те данные проекта, которых у вас ещё нет;
- в рабочем каталоге появятся ссылки на все ветки из удалённого проекта.

Команда `fetch` забирает данные в ваш локальный репозиторий, но не сливает их с какими-либо вашими наработками и не модифицирует то, над чем вы работаете в данный момент. Вам необходимо вручную слить эти данные с вашими, когда вы будете готовы.

pull

git pull - получение данных из удаленного репозитория:

- извлекает (fetch) данные с сервера;
- автоматически пытается слить (merge) их с кодом, над которым вы в данный момент работаете.

push

Когда вы хотите поделиться своими наработками, вам необходимо отправить (push) их в главный репозиторий.

Команда для этого действия простая: `git push [удал. сервер] [ветка]`.

```
$ git push origin master
```

Эта команда срабатывает только в случае, если вы клонировали с сервера, на котором у вас есть права на запись, и если никто другой с тех пор не выполнял команду push.

Если вы и кто-то ещё одновременно клонируете, затем он выполняет команду push, а затем команду push выполняете вы, то ваш push точно будет отклонён.

Вам придётся сначала вытянуть (pull) их изменения и объединить с вашими. Только после этого вам будет позволено выполнить push.

Ветвление в Git

Ветвление

означает, что вы отклоняетесь от основной линии разработки и продолжаете работу, не вмешиваясь в основную линию.

Когда вы фиксируете изменения в Git, Git сохраняет фиксируемый объект, который содержит:

- указатель на снимок содержимого индекса;
- метаданные автора и комментария;
- ноль или больше указателей на коммиты, которые были прямыми предками этого коммита (ноль предков для первого коммита, один — для обычного коммита и несколько — для коммита, полученного в результате слияния двух или более веток).

Пример

Есть каталог, содержащий три файла, они индексируются и делается коммит.

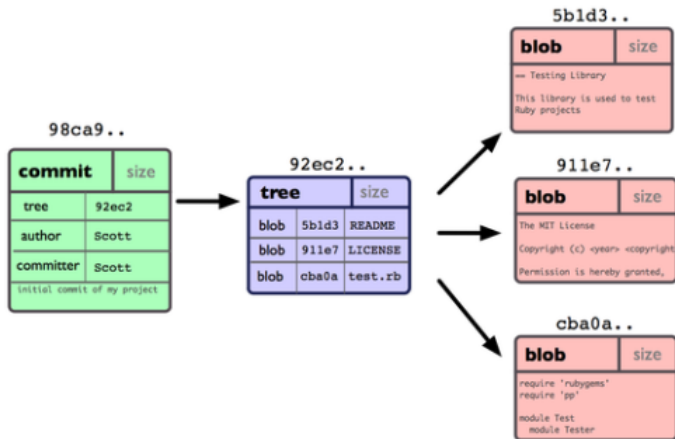
```
$ git add README test.rb LICENSE  
$ git commit -m 'initial commit of my project'
```

При подготовке файлов для каждого из них вычисляется контрольная сумма (SHA-1), затем эти версии файлов сохраняются в Git-репозиторий, а их контрольные суммы добавляются в индекс. При создании коммита Git:

- вычисляет контрольную сумму каждого подкаталога;
- сохраняет объекты для этого дерева в Git-репозиторий;
- создаёт объект для коммита, который имеет метаданные и указатель на корень проектного дерева.

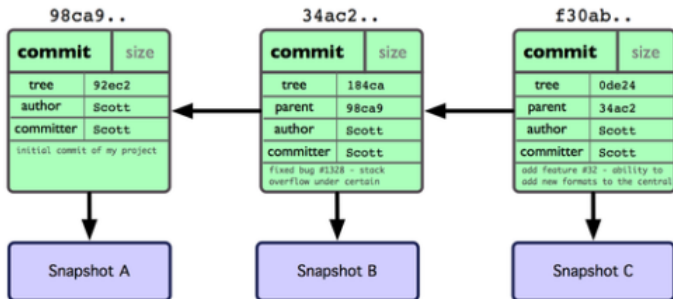
Пример

Данные репозитория с единственным коммитом для трех объектов.



Пример

Данные объектов Git в случае нескольких коммитов.

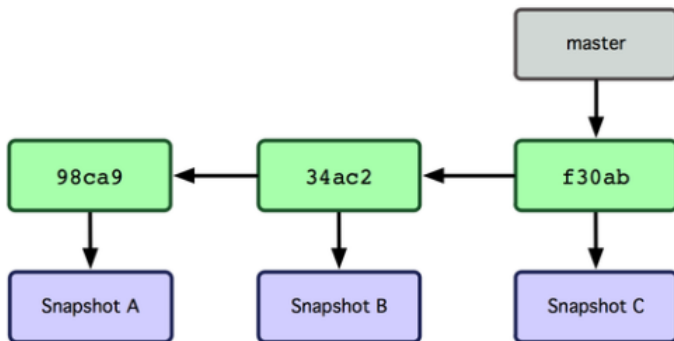


Ветка в Git

Ветка

легковесный подвижный указатель на один из этих коммитов.

Имя ветки по умолчанию в Git — master. При каждом новом коммите указатель сдвигается вперёд автоматически.

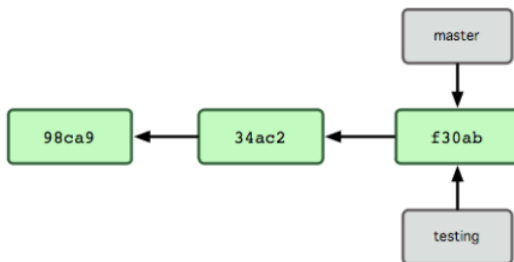


Создание ветки

Создание ветки testing

```
$ git branch testing
```

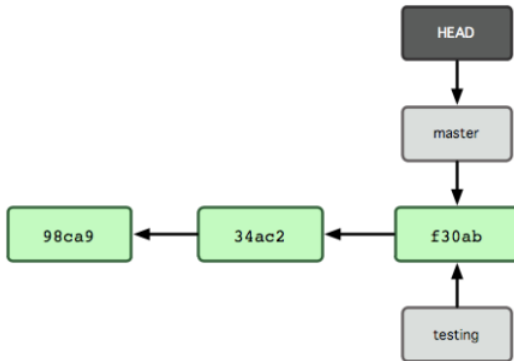
Несколько веток, указывающих на историю коммитов



Указатель на текущую ветку

Git хранит специальный указатель, который называется HEAD (верхушка) - указатель на локальную ветку, на которой вы находитесь.

Файл HEAD указывает на текущую ветку

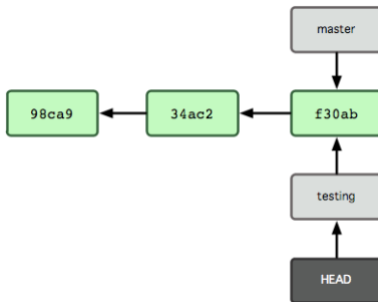


Указатель на текущую ветку

Переход на ветку testing

```
$ git checkout testing
```

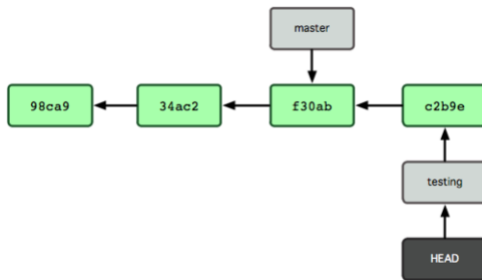
HEAD указывает на другую ветку



Создадим еще один коммит

```
$ vim test.rb  
$ git commit -a -m 'made a change'
```

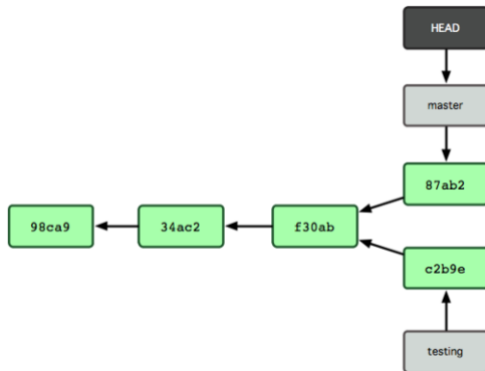
Ветка, на которую указывает HEAD, движется вперед с каждым коммитом



Вернемся в ветку master (git checkout master)

```
$ vim test.rb  
$ git commit -a -m 'made other changes'
```

История с разошедшимися ветками



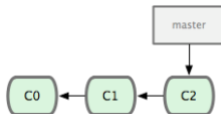
Что нам надо делать?

1. Работать над веб-сайтом.
2. Создадите ветку для новой задачи, над которой вы работаете.
3. Выполните некоторую работу на этой ветке.

... и вдруг

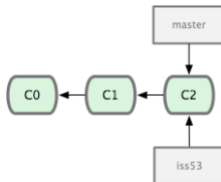
1. Вернётесь на производственную ветку.
2. Создадите ветку для исправления ошибки.
3. После тестирования ветки с исправлением сольёте её обратно и отправите в продакшн.
4. Вернётесь к своей исходной задаче и продолжите работать над ней.

Создаем ветку и переходим в нее



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

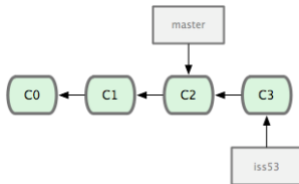
Новая история коммитов



Сделаем пару коммитов

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

Ветка iss53 передвинулась вперёд во время работы



Теперь вы получаете звонок о том, что есть проблема с веб-сайтом, которую необходимо немедленно устранить.

Создадим ветку, в которой будем работать

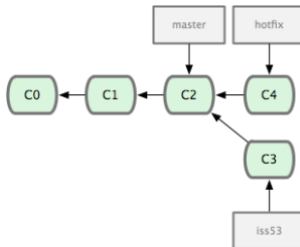
```
$ git checkout master  
Switched to branch "master"
```

Итак, надо срочно исправить ошибку - создадим для этого ветку, на которой вы будете работать.

Ветка для решения срочной проблемы базируется на ветке master

```
$ git checkout -b 'hotfix'  
Switched to a new branch "hotfix"  
$ vim index.html  
$ git commit -a -m 'fixed the broken email address'  
[hotfix]: created 3a0874c: "fixed the broken email address"  
1 files changed, 0 insertions(+), 1 deletions(-)
```

Ветка для решения срочной проблемы базируется на ветке master

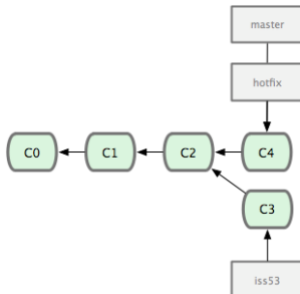


Вы можете запустить тесты, убедиться, что решение работает, и слить (merge) изменения назад в ветку master, чтобы включить его в продукт. Это делается с помощью команды `git merge`.

Сливаем ветки

```
$ git checkout master  
$ git merge hotfix  
Updating f42c576..3a0874c  
Fast forward  
 README |    1 -  
 1 files changed, 0 insertions(+), 1 deletions(-)
```

После слияния ветка master указывает туда же, куда и ветка hotfix



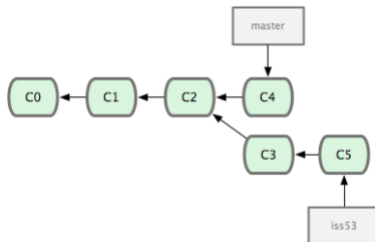
Удаляем ветку hotfix, она больше не нужна

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

Вернемся к нашей исходной задаче

```
$ git checkout iss53  
Switched to branch "iss53"  
$ vim index.html  
$ git commit -a -m 'finished the new footer [issue 53]'  
[iss53]: created ad82d7a: "finished the new footer [issue 53]"  
1 files changed, 1 insertions(+), 0 deletions(-)
```


Ветка `iss53` может двигаться вперёд независимо

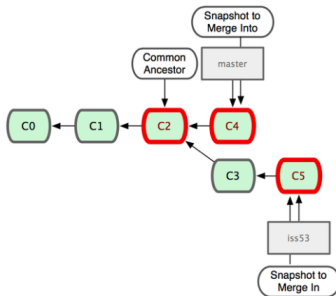


Работа, сделанная на ветке `hotfix`, не включена в файлы на ветке `iss53`. Если вам это необходимо, вы можете выполнить слияние ветки `master` в ветку `iss53` посредством команды `git merge master`.

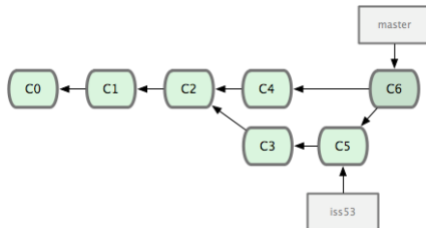
Объединим master и ветку iss53

```
$ git checkout master
$ git merge iss53
Merge made by recursive.
 README |    1 +
  1 files changed, 1 insertions(+), 0 deletions(-)
```

Git автоматически определяет наилучшего общего предка для слияния веток



Git автоматически создает новый коммит, содержащий результаты слияния



и удаляем ненужную ветку

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

Если вы изменили одну и ту же часть файла по-разному в двух ветках, которые собираетесь объединить, Git не сможет сделать это чисто. Если ваше решение проблемы №53 изменяет ту же часть файла, что и hotfix, вы получите конфликт слияния

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

проверяем статус...

```
[master*]$ git status
index.html: needs merge
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
# unmerged:   index.html
#
```

Всё, что имеет отношение к конфликту слияния и что не было разрешено, отмечено как `unmerged`. Git добавляет стандартные маркеры к файлам, которые имеют конфликт, так что вы можете открыть их вручную и разрешить эти конфликты.

```
<<<<<< 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
```

вручную разрешаем конфликт...

```
<div id="footer">
please contact us at email.support@github.com
</div>
```

После того, как вы разрешили каждую из таких секций с каждым из конфликтных файлов, выполните `git add` для каждого конфликтного

Если вы хотите использовать графические инструменты для разрешения конфликтов, можете выполнить команду `git mergetool`, которая запустит соответствующий графический инструмент и покажет конфликтные ситуации:

```
$ git mergetool
merge tool candidates: kdiff3 tkdiff xxdiff meld gvimdiff opendiff emerge vimdiff
Merging the files: index.html
```

