Руководство по Git

Гусев Илья

МФТИ

Москва, 2018

Содержание

VCS

Что такое VCS (система контроля версий)?

Простой сценарий:

1. Создание файла
2. Изменение файла
3. Сохранение файла
4. Goto 2

Обычный сценарий:

1. Создание файла
2. Изменение файла
3. Сохранение файла
4. Изменение файла
5. Сохранение файла
6. Захотелось вернуться в состояние 3

Решается резервным копированием.

VCS

Что такое VCS (система контроля версий)?

Сложный сценарий - 2 человека работают над одним файлом:

1. Создание файла (1 человек)
2. Изменение файла (1 человек)
3. Сохранение файла (1 человек)
4. Получение файла (2 человек)
5. Изменение файла (1 человек)
6. Изменение файла (2 человек)
7. Сохранение файла (2 человек)
8. Получение файла (1 человек), слияние изменений
9. Сохранение файла (1 человек)
10. Получение файла (2 человек)

А ещё может понадобиться возвращение в 3, 7, 9...

VCS

Что такое VCS (система контроля версий)?

Система контроля версий - софт, который позволяет работать в рамках таких сложных сценариев, когда десятки людей вносят изменения в одни и те же файлы. В сочетании с CI (Continuous Integration, непрерывная интеграция) - основа инфраструктуры при разработке почти любого программного продукта.
Можно использовать не только над исходным кодом, а в принципе над любыми текстовыми файлами (и не только текстовыми).

Варианты VCS

Subversion (SVN) vs Git

1. **Централизованная vs распределённая.** У SVN одно хранилище, там лежит полная история, на локальных машинах - последняя версия (working copy). В случае Git'а - каждая машина имеет полную копию репозитория со всей историей. Сервер - такая же машина, только лишь с возможностью pull, push и ограничениями доступа.
2. **Git - ветки**. У Git'а намного более удобная система ветвления. Есть возможность создавать ветки локально (для отдельных фич, например), и потом их целиком сливать на сервер. Ветки очень лёгкие, по сути - всего лишь указатель на коммит.

Git

Установка

1. Скачивание: `https://git-scm.com/downloads`
2. Установка...
3. Первоначальная настройка:

```
git config --global user.name <ваше имя>
git config --global user.email <ваша почта>
git config --global core.editor <ваш любимый редактор>
git config --list
```

Git

Создание или клонирование репозитория

1. Создание репозитория:
   ```
   cd <нужная папка>
   git init
   ```
2. Клонирование репозитория:
   ```
   cd <папка, уровнем выше нужной>
   git clone <адрес репозитория> <имя нужной папки>
   ```

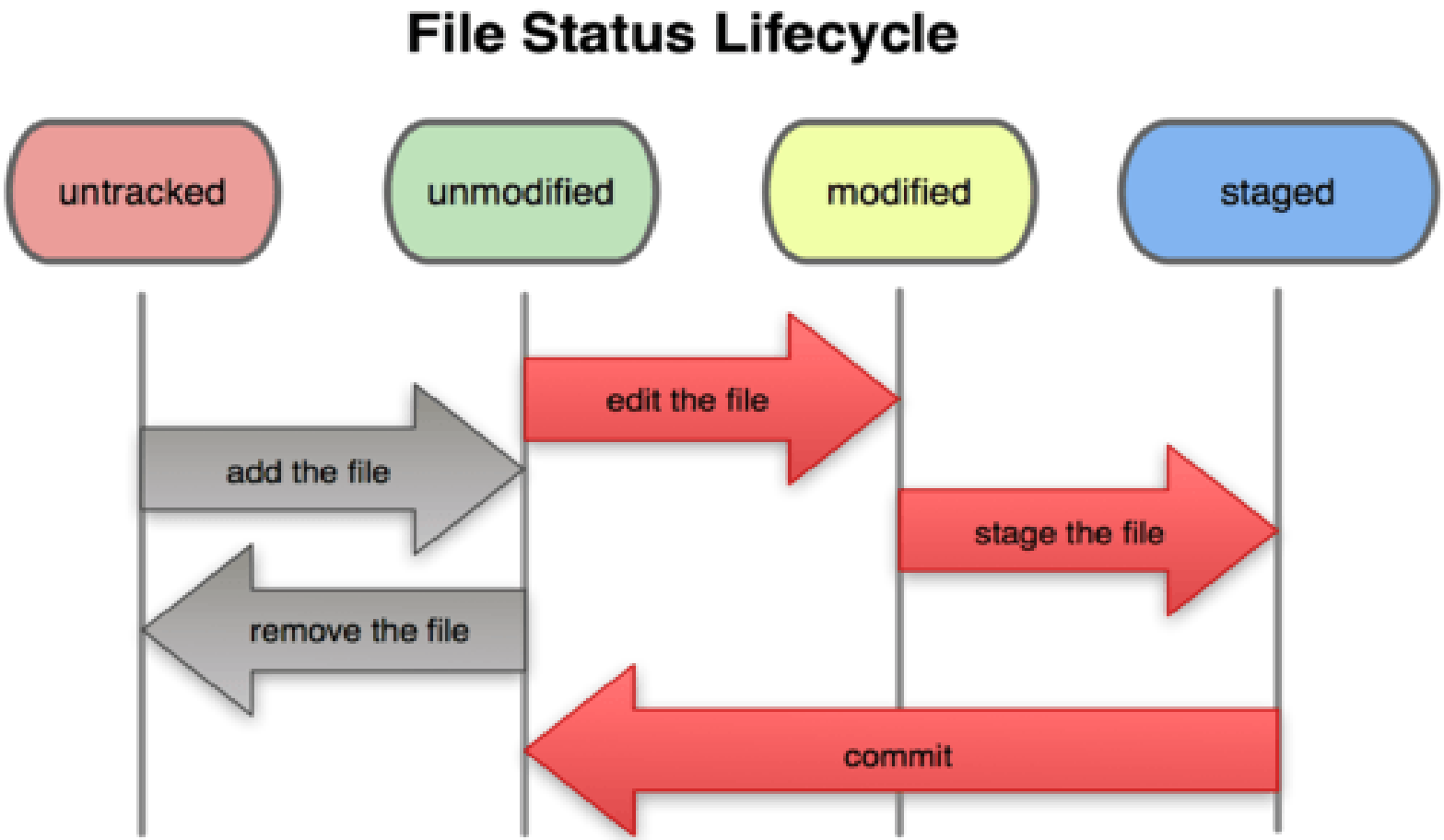
Git

Отслеживание файлов

1. Добавление файла в индекс:
   `git add <имя файла>`
2. Добавление всех файлов в индекс:
   `git add -A`
3. Удаление файла из отслеживаемых:
   `git rm <имя файла>`
4. Фиксация изменений
   `git commit -m "Сообщение при коммите"`

Git

Отслеживание файлов



Git

Отслеживание файлов

1. Интерактивное добавление:
   `git add -i`
2. Добавление по частям
   `git add -p`
3. Добавление без новых файлов
   `git add -u`

Git

.gitignore

Файл в корне репозитория, определяет, какие файлы автоматически игнорируются. Пример из git-book:

```
# комментарий - эта строка игнорируется
# не обрабатывать файлы, имя которых заканчивается на .a
*.a
# НО отслеживать файл lib.a, несмотря на то, что мы игнорируем все .
!lib.a
# игнорировать только файл TODO находящийся в корневом каталоге, не
/TODO
# игнорировать все файлы в каталоге build/
build/
# игнорировать doc/notes.txt, но не doc/server/arch.txt
doc/*.txt
# игнорировать все .txt файлы в каталоге doc/
doc/**/*.txt
```

Git

Отмена изменений

1. Отмена последнего коммита с сохранением изменений в индексе:
   `git reset --soft HEAD~1`
2. Отмена последнего коммита без сохранения изменений в индексе, но с сохранением изменений в рабочей копии:
   `git reset --mixed HEAD~1`
3. Отмена всех изменений (hard reset) до последнего коммита:
   `git reset --hard HEAD`
4. Отмена всех изменений в файле до последнего коммита:
   `git checkout -- <имя файла>`
5. Отмена всех изменений в файле до какого-то коммита:
   `git checkout <коммит> -- <имя файла>`
6. Отмена всех изменений (hard reset) до N-ого коммита с конца:
   `git reset --hard HEAD~N`
7. Удаление неотслеживаемых фалов и папок:
   `git clean -fd`

Git

Отмена изменений

1. Исправление последнего коммита:
   ```
   git commit --amend
   ```
   Пример:
   ```
   git commit -m "Сообщение при коммите"
   git add <забытый файл>
   git commit --amend
   ```

Git

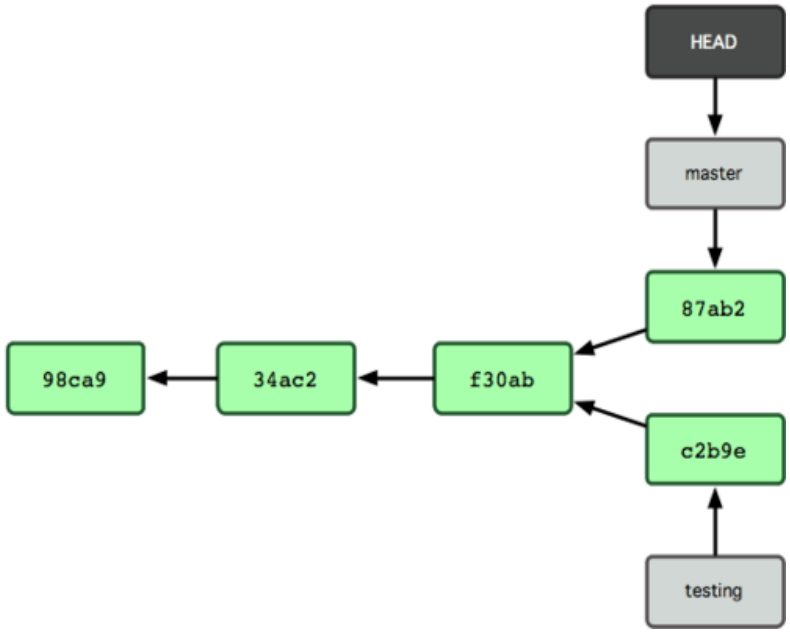
Служебные команды

1. Лог коммитов:
   `git log`
2. Лог коммитов (красивый, с веточками):
   `git log --graph`
3. Лог коммитов (один коммит - одна строчка):
   `git log --pretty=oneline`
4. Статус индекса:
   `git status`
5. Просмотр изменений:
   `git diff`
6. Просмотр индексированных изменений:
   `git diff --cached`
7. Сжатие (происходит автоматически при push):
   `git gc`

Git

Ветки

1. Каждый коммит характеризуется несколькими основными вещами: хеш (на картинке - его первые 5 цифр), автор, дата.
2. HEAD - метка показывающая на текущий коммит, который мы сейчас изменяем.
3. Ветка - формально, указатель на коммит. Иногда ещё подразумевают всю историю до этого коммита.



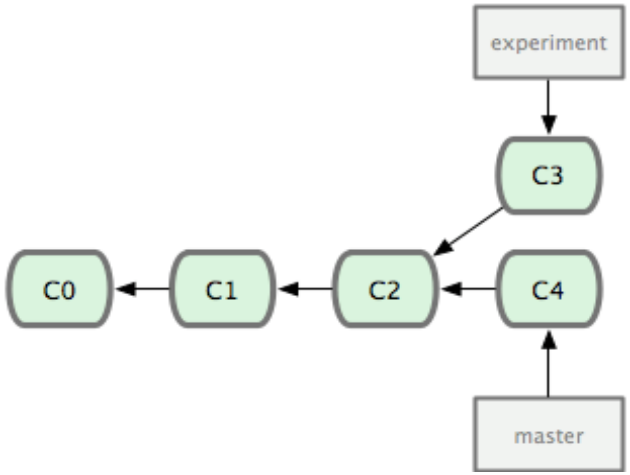
Git

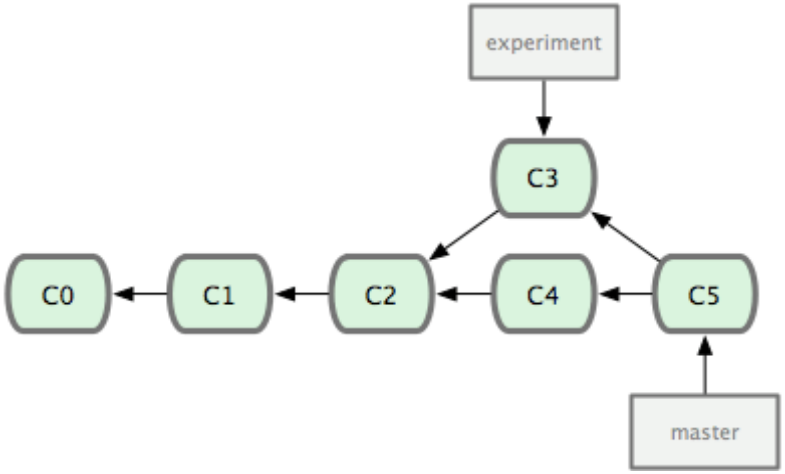
Операции с ветками

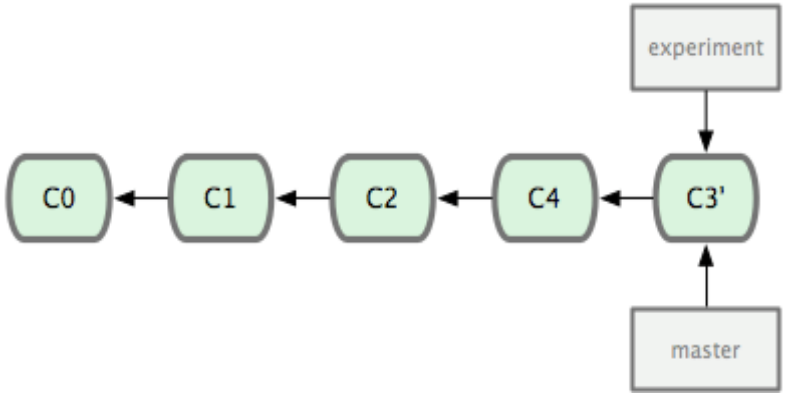
1. Создание ветки (указывает на текущий коммит):
   `git branch <название ветки>`
2. Переход на ветку:
   `git checkout <название ветки>`
3. Слияние:
   `git merge <из какой ветки> <в какую ветку>`
4. Слияние одним коммитом:
   `git merge --squash <из какой ветки> <в какую ветку>`
5. Перемещение:
   `git rebase <из какой ветки> <в какую ветку>`
6. При успешном разрешении конфликтов слияния:
   `git commit -a`

Git

Слияние vs перемещение



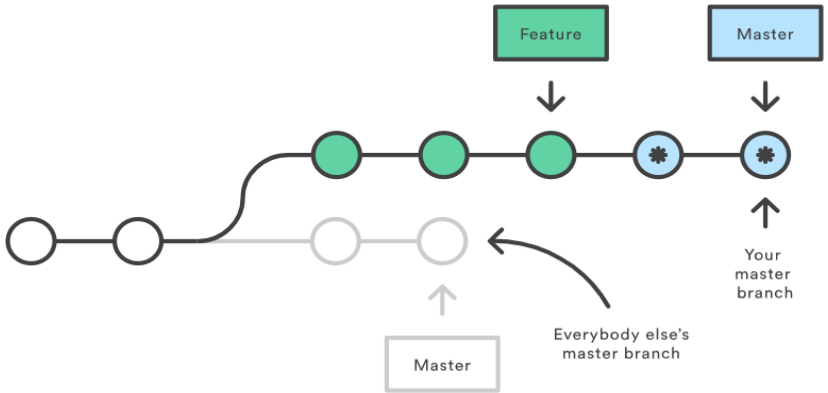




Git

Слияние vs перемещение

1. Merge - безопасный, но создаётся лишний коммит. Не меняет существующие коммиты.
2. Rebase - переписывает историю, для каждого коммита из вливаемой ветки создаётся новый коммит с новым хешом (так как файлы уже в другом состоянии). Небезопасно, но получившаяся история линейна. Теряем информацию, что когда-то это была отдельная ветка.
3. Не стоит использовать rebase на публичных ветках из-за того, что он перезаписывает историю и меняет хеши коммитов.



Git

Удалённые репозитории

1. Добавление удалённого репозитория:
   `git remote add <сокращение> <url>`
2. Получение новых изменений в копии удалённых веток:
   `git fetch`
3. Получение новых изменений (с автослиянием в локальные ветки):
   `git pull`
4. Получение новых изменений из конкретной ветки (с автослиянием)
   `git pull <имя репозитория> <имя ветки>`
5. Pull выделили чисто для удобства. Он может значить разное в зависимости от настроек (либо `fetch & merge`, либо `fetch & rebase`)
6. Отправка изменений:
   `git push <имя репозитория> <имя локальной ветки>:<имя удалённой ветки>`
7. Отслеживание удалённой ветки
   `git checkout --track <имя репо>/<имя ветки>`

Git

Stash

1. Временно сохранить текущие изменения:
   `git stash`
2. Восстановить последние сохранённые изменения и удалить их из списка:
   `git stash pop`
3. Список припрятанных изменений:
   `git stash list`
4. Применить конкретные припрятанные изменения:
   `git stash apply <id изменений>`
5. Создать ветку со спрятанными изменениями, причём с началом в том коммите, где прятали:
   `git stash branch <имя ветки>`

Git

Элиасы

1. git config —global alias.<название элиаса> <команда гита>
2. Примеры:
   1. git config —global alias.co checkout
   2. git config —global alias.ci commit
3. Либо в .gitconfig:
   [*alias*]
   co = checkout
   ci = commit

Git

Восстановление данных

1. Каждый раз при переключении веток и коммите, добавляется запись в reflog вида:
   `1a410ef HEAD@{0}: 1a410efbd13591db07496601ebc7a059dd55cfe9: updating HEAD`
2. Список записей:
   `git reflog --all`
3. Проверка на целостность с выводом недоступных объектов:
   `git fsck --full`

Workflow

Репозиторий

1. Сервер - Bitbucket.
2. Создаёте приватный репозиторий.
3. Добавляете семинаристов (Settings -> User and group access).
4. Даёте админские права семинаристам.
5. Вся работа в рамках этого репозитория.
6. Ревью проходит в пулл-реквестах на Bitbucket.
7. Не забудьте про .gitignore!
8. Инструкция с картинками по клику

Workflow

Устройство веток в вашем репозитории

1. master - ветка, куда вливается подтверждённый семинаристом код.
2. Для каждого ревью отдельная ветка. Стандартный сценарий при настроенном репозитории:
   1. `git checkout master`
   2. `git checkout -b review-<номер ревью>`
   3. `делаете изменения`
   4. `git commit -a -m <сообщение при коммите, какое угодно>`
   5. `git push <имя репозитория> review-<номер ревью>`

Workflow

Пулл-реквесты

1. Пулл-реквесты делаются в веб-интерфейсе Bitbucket.
2. Когда всё готово, делаете pull request из вестки review-<номер ревью> в master. Названия pull request'ов должны быть по шаблону:
   `[Имя Фамилия] Ревью-<номер ревью>`
3. Семинаристы делают ревью, вы исправляете замечания.

Полезные ссылки I

Pro Git
https://git-scm.com/book/ru/v1