Руководство по Git

Гусев Илья

Московский физико-технический институт

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)?

- Программа для отслеживания изменений в файлах
- Необходима, когда код редактирует больше 1 человека
- Полезна, когда код редактирует даже 1 человек
- Полезна не только в случае кода

Понятия

- Репозиторий (репо) - набор файлов и истории их изменений
- Ветка - набор конкретных изменений
- Коммит - фиксированное и сохранённое состояние файлов
- Закоммитить - сделать какие-то изменения и зафиксировать их
- Запушить - отправить коммит на удалённый сервер
- Откатить - перенестись на какой-то старый коммит

Варинты 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.
3. Первый выпуск: 2000 vs 2005.

Не путайте!

Git vs GitHub

- Git - система контроля версий, консольная утилита
- GitHub - сайт, на котором можно хранить репозитории Git'а
- Bitbucket - другой такой сайт с бесплатной возможностью создания закрытых репо

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 индекса:
   `git rm --cached <имя файла>`
4. Фиксация изменений
   `git commit -m "Сообщение при коммите"`
5. Добавление всех файлов + фиксация
   `git commit -a -m "Сообщение при коммите"`

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 .a файлы с
    помощью предыдущего правила
!lib.a
# игнорировать только файл TODO находящийся в корневом каталоге, не относится к
    файлам вида subdir/TODO
/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. Исправление последнего коммита:
   `git commit --amend`
   Пример:
   `git commit -m "Сообщение при коммите"`
   `git add <забытый файл>`
   `git commit --amend`
2. Отмена индексации (soft reset) до последнего коммита:
   `git reset --soft HEAD`
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 (hard reset) до N-ого коммита с конца:
   `git reset --hard HEAD~N`

Git

Служебные команды

1. Лог коммитов:
   `git log`
2. Лог коммитов (красивый, с веточками):
   `git log --graph`
3. Статус индекса:
   `git status`
4. Просмотр изменений:
   `git diff`
5. Просмотр индексирвоанных изменений:
   `git diff --cached`
6. Сжатие (происходит автоматически при push):
   `git gc`

Git

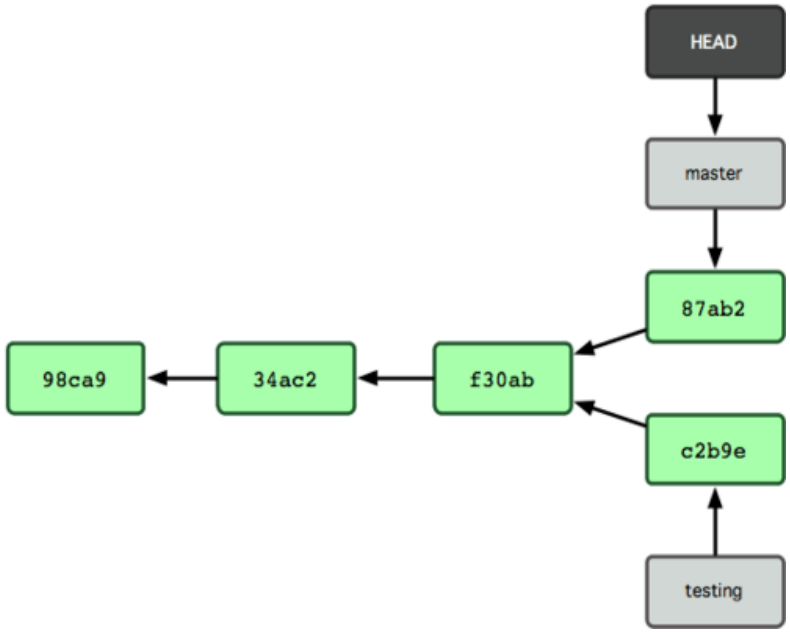
Удалённые репозитории

1. Добавление удалённого репозитория:
   `git remote add <сокращение> <url>`
2. Получение новых изменений:
   `git fetch`
3. Получение новых изменений (с автослиянием):
   `git pull`
4. Получение новых изменений из конкретной ветки (с автослиянием)
   `git pull <имя репозитория> <имя ветки>`
5. Отправка изменений:
   `git push <имя репозитория> <имя ветки>`

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>`
4. При успешном разрешении конфликтов слияния:
   `git commit -a`

Workflow

Репозиторий

1. Сервер - Bitbucket.
2. Создаёте приватный репозиторий.
3. Добавляете семинаристов (Settings -> User and group access).
4. Даёте админские права.
5. Вся работа в рамках этого репозитория.
6. Без пройденного ревью на Bitbucket задача не засчитывается!
7. Не забудьте про .gitignore!

Workflow

Устройство веток в вашем репозитории

1. master - ветка, куда вливается подтверждённый семинаристом код.
2. Для каждой задачи отдельная ветка. Стандартный сценарий при настроенном репозитории:
   1. `git checkout master`
   2. `git checkout -b <название или номер задачи>`
   3. `делаете изменения, сдаёте в контесте`
   4. `git commit -a -m <сообщение>`
   5. `git push <имя репозитория> <имя ветки>`

Workflow

Устройство веток в вашем репозитории

1. Когда всё готово, делаете pull request в master. Названия pull request'ов должны быть по шаблону:
   `[Имя Фамилия] ДЗ-<номер ДЗ>, <название задачи>.`
2. В описании pull request'а должна быть ссылка на успешную посылку в контесте.
3. Семинаристы делают ревью, вы исправляете замечания.

Полезные ссылки I

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