Занятие 1. git

Гирдюк Дмитрий Викторович

14 сентября 2024

СПбГУ, ПМ–ПУ, ДФС

git: история

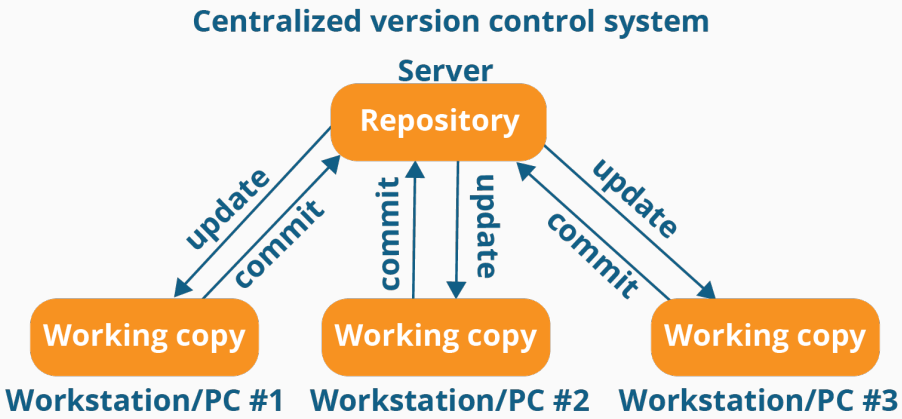
- Разработан Линусом Торвальдсом, создателем Linux, в 2005 г. Изначально предназначался для управления версиями ядра Linux.
- Основные цели разработки git:
  - Чтобы уметь в организацию огромных проектов + скорость.
  - Поддержка многопользовательской разработки (тыщи параллельных веточек) + полностью распределенная.
- Прямо-таки откровение: “git” is an unpleasant or contemptible person.

Установка и ссылочки

- Официальный сайт: `http://git-scm.com/`.
- Pro git: `http://git-scm.com/book`.
- “git for computer scientists”: `http://eagain.net/articles/git-for-computer-scientists/`.
- git help command.

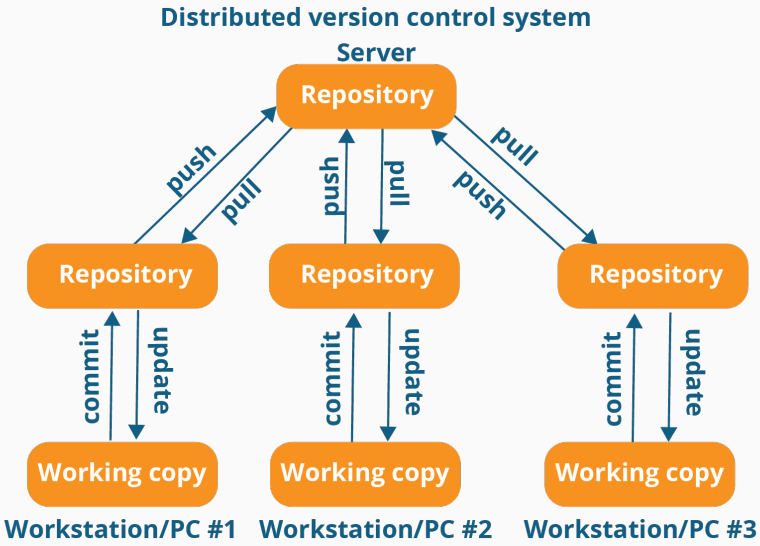
Централизованная система контроля версий

- В Subversion, CVS, Perforce и т.д. репозиторий центрального сервера хранит “официальную копию” кода – сервер обслуживает единственную история версий репозитория.
- При работе производится “checkout” в вашу локальную копию репозитория. Внесенные локальные изменения не версионируются.
- Завершая работу, происходит “check in” – регистрация изменений на сервере, что приводит к появлению новой версии.



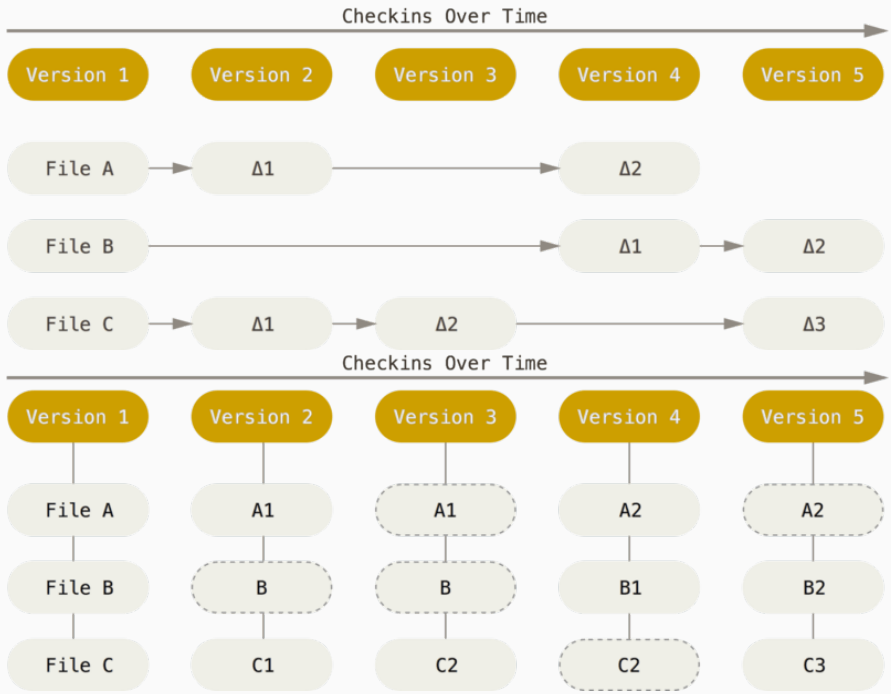
Распределенная система контроля версий

- В git, mercurial и т.д. не нужно делать “checkout” изменений из централизованного репозитория
  - Происходит clone репозитория на локальную машину и, таким образом, pull изменений
- Большинство операций локальны
  - checkout в рамках локального репозитория
  - commit изменений в локальном репозитории
  - локальный репозиторий хранит всю историю изменений
- При завершении работы изменения “push”-атся обратно на удаленный сервер.



Как работает?

- Большинство других систем хранят информацию в виде списка файловых изменений. Эти системы рассматривают информацию, которую они хранят, как набор файлов и изменения, вносимые в них с течением времени.
- С git'ом все иначе. git рассматривает данные как серию "снимков" (snapshots) собственной файловой системы.



Как работает?

- По своей сути git есть ничто иное как миниатюрная файловая система с кучей удобных утилит, а не просто “одна из” VCS.
- Еще раз, все взаимодействия локальны, не требуют доступа у удаленному репозиторию после его клонирования. Это позволяет получать доступ к любому участку истории репозитория.
- И еще одна важная особенность: git умеет в целостность данных. Перед загрузкой данных все хэшируется (SHA-1), потому нет возможности изменить какой-либо файл так, чтобы git не узнал об этом.

Три основные состояния

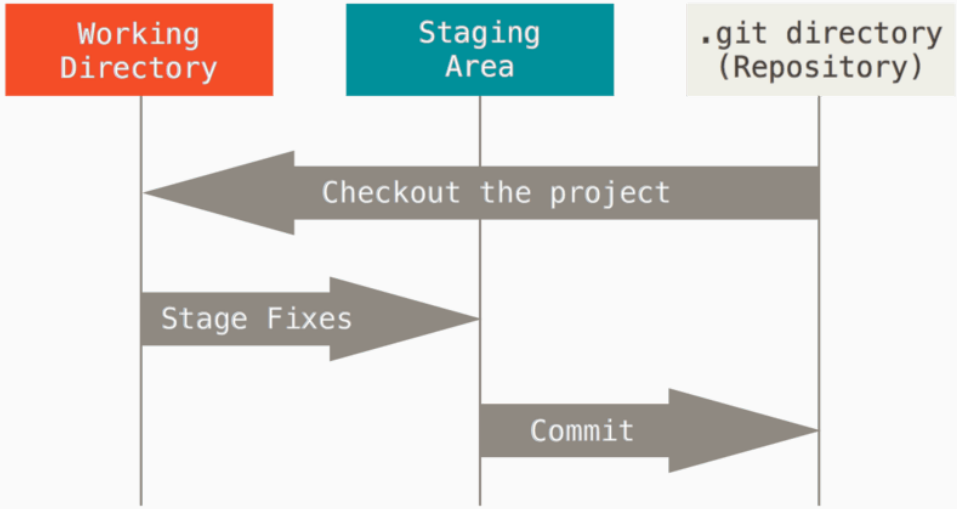
В локальной копии репозитория файлы могут находиться в трех состояниях

- Рабочая директория. Эти файлы извлекаются из сжатой базы данных в каталоге git и помещаются на диск.
- Index/staging area – это файл, обычно содержащийся в каталоге git, в котором хранится информация о том, что войдет в вашу следующую версию.
- Директория git'a (git directory/repo). git хранит метаданные и базу данных объектов для вашего проекта. Это самая важная часть git, и это то, что копируется, когда происходит клонирование репозитория с удаленного сервера/другого компьютера.

Рабочий процесс

Базовый рабочий процесс с git выглядит следующим образом

- Производятся какие-либо изменения в рабочей директории.
- Выборочно обрабатываете только те изменения, которые хотите включить в следующий commit, который добавляет только эти изменения в index.
- Выполняете commit, который фиксирует файлы такими, как они находятся в индексе, и сохраняет этот snapshot в каталоге git. Каждый commit хэшируется, доступ к нему можно получить, зная его хэш.



Предварительная конфигурация

- Задайте имя и адрес электронной почты, которые git будет использовать при commit'e:
  - git config −−global user.name "Dmitry Girdyuk"
  - git config −−global user.email dm.girdyuk@gmail.com
  - Проверка: git config −−list
- Конфигурация текстового редактора для написания сообщений, описывающих commit (vim по умолчанию)
  - git config −−global core.editor nano

Создание, запись и контроль

- git init – создание нового репозитория
- git add [−A] file1 file2 dir – добавление внесенных изменений в индекс
- > .gitignore – список файлов и директорий, которые git следует полностью игнорировать
- git diff [−−staged/cached] – показывает разницу между тем, что в индексе и тем, что изменено, но еще в него не добавлено (−−staged – сравнение с последним коммитом)
- git commit −m “описание commit’a”
- git status
- git rm [−−cached]
- git mv <file_from> <file_to> (mv -> git rm -> git add)

Отмена изменений

- git commit −−amend – изменение последнего коммита
- git reset HEAD <file_name> – вытащить из индекса
- git checkout HEAD/<commit_hash> <file_name> – удалить все изменения в файле относительно предыдущего/некоторого commit'a

Просмотр истории

git log – показывает в обратном хронологическом порядке все коммиты

- −p/−−patch – показывает еще и изменения (или их часть)
- −−stat
- −−pretty=...
  - oneline
  - format: %H, %h, %T, %t, %P, %p, %an, %ae, %ad, %cn, %ce, %cd, %cr, %s
- −−graph – отображает граф истории ветвления и слияния рядом с log'ом.
- −−since, −−until

Тэги

- git tag – показать список текущих тегов
  - git tag <tag_name>/−a <tag_name> −m “message” [commit hash] – создание тега
  - git show <tag_name>
  - git tag −d <tag_name> – удаление тэга

Ветвление и слияние

- git branch [−v | −vv, −−merged, −−no-merged] – просмотр всех веток (по своей сути ветка – это просто указатель на коммит)
- git checkout -b <branch_name> – oneliner
- git checkout <branch_name> – переход на ветку
- git branch -d <branch_name> – удаление ветки
- git merge <branch_name> – слияние текущей ветки и ветки branch_name
- git rebase <branch_name>/master – перебазирование коммитов с текущей ветки на master/branch_name
- HEAD – указатель на текущую ветку, которая указывает на текущий коммит

Подробнее про reset

- git reset [−−soft | −−mixed | −−hard] HEAD∼/<commit_name> – перемещает ветку, на которую указывает HEAD на произвольный коммит
  - −−soft – просто перемещает ветку
  - −−mixed (по умолчанию) – перемещает ветку и чистит индекс
  - −−hard – приводит рабочий каталог к тому же виду, что и индекс
- git reset −−hard HEAD∼ – откат слияния (в рамках локального репозитория)

Исправление истории

- git commit ––amend – изменение последнего коммита
- git rebase –i HEAD~k – правки последних k коммитов. В открывшемся файле
  - переупорядочивание коммитов
  - pick – ничего не делаем, коммит остается в исходном виде
  - reword – меняем лишь сообщение в коммите
  - edit – тут можно делать git commit ––amend, да и возможно разбиение коммита путем чистки индекса командой git reset HEAD~
  - drop – удаляем коммит (можно просто удалить строку с коммитом)
  - squash – объединение с предыдущем коммитом
  - git rebase ––continue/––abort
- git filter-branch

Работа с удаленными репозиториями

- git clone <url> – получение копии существующего git-репозитория
- git remote [−v] – просмотр настроенных удаленных репозиториев (origin – имя по умолчанию для сервера наравне с веткой master)
  - git remote add <short_name> <url> – добавление нового удаленного репозитория + присвоение имени
  - git remote rename origin originalniy
  - git remote rm bug01

Работа с удаленными репозиториями

- git fetch [−−all | <short_name>] – получение данных из origin (short_name) без слияния
  - git checkout −b tmp origin/tmp | git checkout −−track origin/tmp | git checkout tmp – чтобы иметь возможность изменять что-либо в ветке tmp и пушить изменения на сервер
- git pull [<short_name>] – fetch + merge
- git push <remote_name> <branch_name> – отправка изменений в удаленный репозиторий
  - git push origin −−delete tmp