Міністерство освіти та науки України
Національний технічний університет України
«Київський політехнічний інститут ім. І.Сікорського»
Навчально-науковий комплекс
«Інститут прикладного системного аналізу»

Лабораторна робота №1
З курсу *«ПРОЕКТУВАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ»*
Тема: *«Системи контролю версій SVN, GIT.»*

Варіант №7

Виконав:
Студент 4-го курсу
Групи ДА-61
Грудович В.І.

Київ 2019

Мета роботи: за допомогою системи контролю версій завантажити коди програми у репозиторій. Відтворити типовий цикл розробки програмного забезпечення з використанням системи контролю версій.
Задача:
1. Вивчити основні команди роботи з репозіторіями.
2. Завантажити код програми у репозіторій.
3. Показати основний цикл роботи з програмним кодом за допомогою системи контролю версій.

Завдання

1. Обрати безкоштовну систему репозиторія для системи контролю версіями, наприклад projectlocker, або інш.
2. Встановити клієнтське безкоштовне програмне забезпечення для роботи с системою контролю версій (GIT, SVN clients).
3. Протягом роботи над лабораторними роботами 2-6 використовувати систему контролю версіями.
4. Описати цикл розробки програмного забезпечення з використанням системи контрою версій.

Короткі теоретичні відомості

Система керування версіями (від англ Version Control System, VCS або Revision Control System.) – це спеціальне програмне забезпечення для полегшення роботи з інформацією, яка часто змінюється. Система керування версіями дозволяє зберігати декілька версій одного і того ж документа, при необхідності повертатися до більш ранніх версій, визначати, хто і коли зробив ту чи іншу зміну, керувати гілками різноманітних версій програми.

GIT

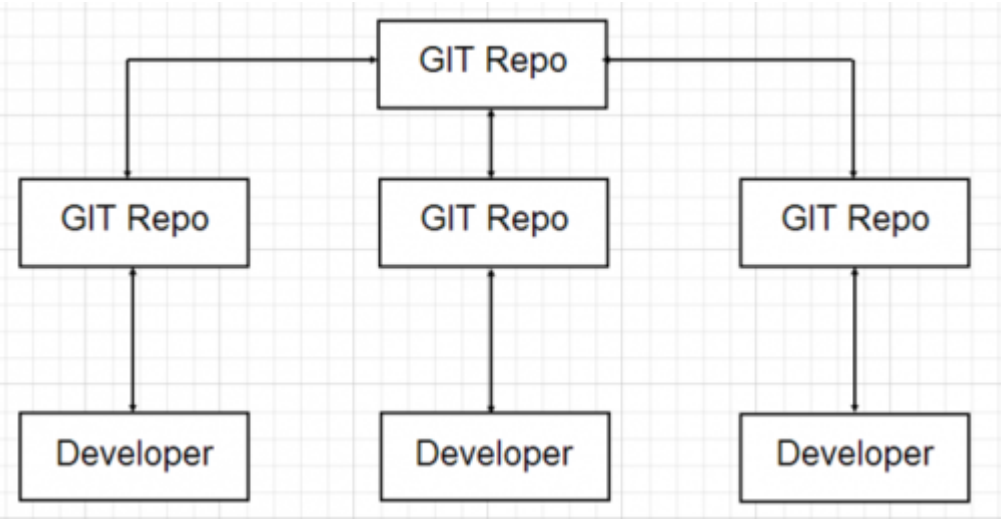
Git (произносится «гит») — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года.

Особенности реализации

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и даёт возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом).

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранящие операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы. Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создаёт в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «.git» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий git, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети. При импорте нового репозитория автоматически создаётся рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда commit).

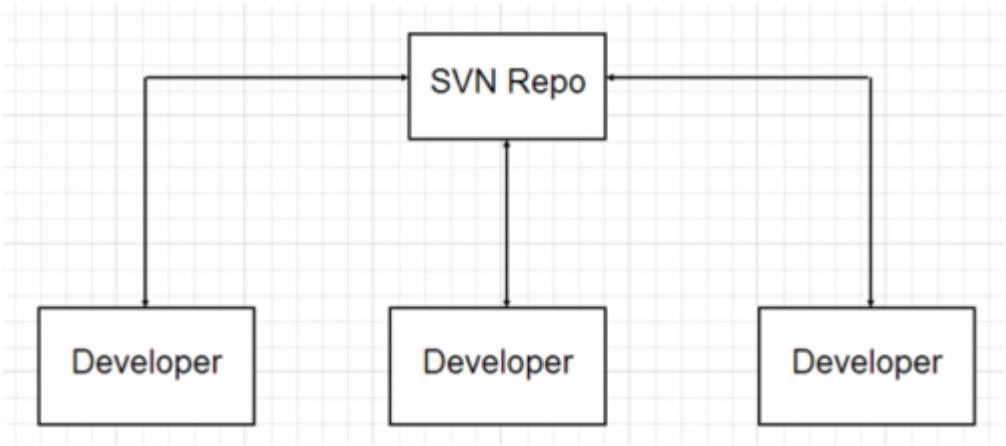


SVN

Subversion(также известная как «**SVN**») — свободная централизованная система управления версиями, официально выпущенная в 2004 году компанией CollabNet Subversion — централизованная система (в отличие от распределённых систем, таких как Git или Mercurial), то есть данные хранятся в едином хранилище. Хранилище может располагаться на локальном диске или на сетевом сервере.

Работа в Subversion мало отличается от работы в других централизованных системах управления версиями. Клиенты копируют файлы из хранилища, создавая локальные рабочие копии, затем вносят изменения в рабочие копии и фиксируют эти изменения в хранилище. Несколько клиентов могут одновременно обращаться к хранилищу. Для совместной работы над файлами в Subversion преимущественно используется модель *копирование — изменение — слияние*. Кроме того, для файлов, не допускающих слияние (различные бинарные форматы файлов), можно использовать модель *блокирование — изменение — разблокирование*.

При сохранении новых версий используется дельта-компрессия: система находит отличия новой версии от предыдущей и записывает только их, избегая дублирования данных.



GIT vs SVN

- GIT распределяется, а SVN - нет. Другими словами, если есть несколько разработчиков работающих с репозиторием у каждого на локальной машине будет ПОЛНАЯ копия этого репозитория. Разумеется есть и где-то и центральная машина, с которой можно клонировать репозиторий. Это

напоминает SVN. Основной плюс в том, что если вдруг у вас нет доступа к интернету, сохраняется возможность работать с репозиторием. Потом только один раз сделать синхронизацию и все остальные разработчики получат поолную историю.

- GIT сохраняет метаданные изменений, а SVN целые файлы. Это экономит место и время.
- Система создания branches, versions и прочее в GIT и SVN отличаются значительно. В GIT проще переключатся с ветки на ветку, делать merge между ними.
- SVN позволяет проверять только субдерева (или ветви), тогда как Git требует от вас проверить весь хранилище как единицу. Это происходит потому, что в каждой из ваших папок является .svn, а в git есть только одна .git в родительском каталоге верхнего уровня

Основные команды GIT

Если вы собираетесь начать использовать Git для существующего проекта, то вам необходимо перейти в проектный каталог и в командной строке ввести

```
$ git init
```

Эта команда создаёт в текущем каталоге новый подкаталог с именем .git содержащий все необходимые файлы репозитория — основу Git-репозитория. На этом этапе ваш проект ещё не находится под версионным контролем.

Если вы хотите добавить под версионный контроль существующие файлы (в отличие от пустого каталога), вам стоит проиндексировать эти файлы и осуществить первую фиксацию изменений. Осуществить это вы можете с помощью нескольких команд `git add` указывающих индексируемые файлы, а затем `commit`:

```
$ git add *.c
```

```
$ git add README
```

```
$ git commit -m 'initial project version'
```

Клонирование существующего репозитория

Клонирование репозитория осуществляется командой `git clone [url]`. Например, если вы хотите клонировать библиотеку Ruby Git, известную как Grit, вы можете сделать это следующим образом:

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

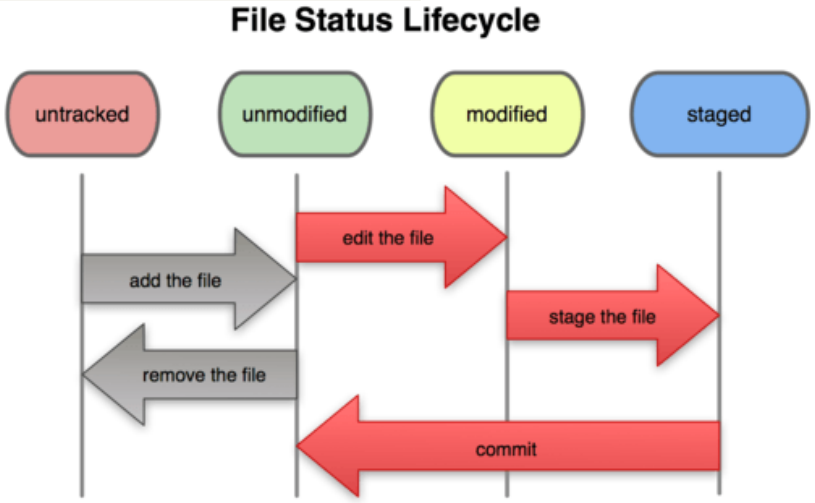


Рисунок 2-1. Жизненный цикл состояний файлов.

Определение состояния файлов

Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся — это команда `git status`. Если вы выполните эту команду сразу после клонирования, вы увидите что-то вроде этого:

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

Просмотр истории коммитов

После того как вы создадите несколько коммитов, или же вы склонируете репозиторий с уже существующей историей коммитов, вы, вероятно, захотите оглянуться назад и узнать, что же происходило с этим репозиторием. Наиболее простой и в то же время мощный инструмент для этого — команда `git log`.

```
$ 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 a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit
```

git diff

Чтобы увидеть, что же вы изменили, но пока не проиндексировали, наберите **git diff** без аргументов

git diff --stat

Показывает, как каждый файл был изменен за определенное время. Вы можете добавить 3 параметра: **width** для определения ширины вывода по умолчанию, **name-width** для установки ширины имени файла и **count** для ограничения вывода на первое число строк.

```
$ git diff --stat
 index.php | 83 +++++++++++++++++++++++++++++++++++++++++++-----------------------------
 1 file changed, 43 insertions(+), 40 deletions(-)
$ git diff --stat-width=10
 index.php | 83 +++---
 1 file changed, 43 insertions(+), 40 deletions(-)
```

git pull --rebase

Git pull --rebase заставляет git сначала вытащить изменения, а затем переустановить разблокированные коммиты поверх последней версии удаленной ветки. Параметр --rebase может использоваться для создания линейной истории, избегая ненужных фиксаций.

git fetch

Команда git fetch связывается с удалённым репозиторием и забирает из него все изменения, которых у вас пока нет и сохраняет их локально.

git pull

Команда git pull работает как комбинация команд git fetch и git merge, т.е. Git вначале забирает изменения из указанного удалённого репозитория, а затем пытается слить их с текущей веткой.

git push

Команда git push используется для установления связи с удалённым репозиторием, вычисления локальных изменений отсутствующих в нём, и собственно их передачи в вышеупомянутый репозиторий. Этой команде нужно право на запись в репозиторий, поэтому она использует аутентификацию.

git merge

Команда git merge используется для слияния одной или нескольких веток в текущую. Затем она устанавливает указатель текущей ветки на результирующий коммит.