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 №25*

Виконав:

Студент 4-го курсу

Групи ДА-61

*Халета М. В.*

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.

**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.

**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.

**Зміст звіту**

1. Мета роботи.

2. Завдання роботи.

3. Оформлення результатів роботи:

3.1. Опис команд, які використовувалися протягом виконання роботи з системою контролю версіями.

3.2. Лістинг каталогів у репозиторію для програмних кодів для л.р. 2-6.

4. Висновки.

**Контрольні питання**

1. Які основні команди для роботи з системою контролю версій ви знаєте?

2. Опишіть цикл розробки програмного забезпечення з використанням системи контролю версій?

3. Які типові помилки трапляються при роботі з системою контролю версій під час розробки програмного продукту? Як уникнути таких помилок?

1. Обрати безкоштовну систему репозиторія для системи контролю версіями, наприклад projectlocker, або інш.

Для виконання лабораторних робіт було вибрано систему контролю версій Github.

Було створено репозиторій для половини студентів группи, у якому є дві гілки: dev and master. В гільці dev буде проводись вся робота.



2. Встановити клієнтське безкоштовне програмне забезпечення для роботи с системою контролю версій (GIT, SVN clients).

Було встановлено системою контролю версій – GIT.



3.1. Опис команд, які використовувалися протягом виконання роботи з системою контролю версіями.

[**Создание репозитория в существующем каталоге**](#Создание-репозитория-в-существующем-кат)

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. (В главе 9 приведено подробное описание файлов содержащихся в только что созданном вами каталоге .git)

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-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

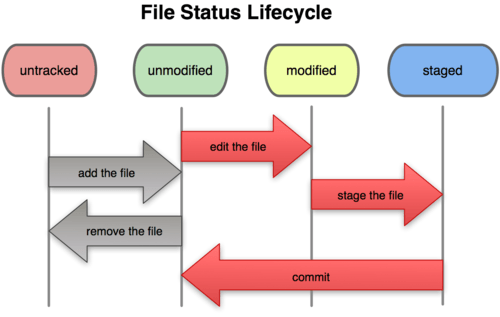


Рисунок 2-1. Жизненный цикл состояний файлов.

### [Определение состояния файлов](" \l "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.

Данные примеры используют очень простой проект, названный simplegit, который я часто использую для демонстраций. Чтобы получить этот проект, выполните:

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

В результате выполнения git log в данном проекте, вы должны получить что-то вроде этого:

$ 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

По умолчанию, без аргументов, git log выводит список коммитов созданных в данном репозитории в обратном хронологическом порядке. То есть самые последние коммиты показываются первыми. Как вы можете видеть, эта команда отображает каждый коммит вместе с его контрольной суммой SHA-1, именем и электронной почтой автора, датой создания и комментарием.

### [Индексация изменённых файлов](#Индексация-изменённых-файлов)

Давайте модифицируем файл, уже находящийся под версионным контролем. Если вы измените отслеживаемый файл benchmarks.rb и после этого снова выполните команду status, то результат будет примерно следующим:

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# new file: README

#

# Changes not staged for commit:

# (use "git add <file>..." to update what will be committed)

#

# modified: benchmarks.rb

#

Файл benchmarks.rb находится в секции “Changes not staged for commit” — это означает, что отслеживаемый файл был изменён в рабочем каталоге, но пока не проиндексирован. Чтобы проиндексировать его, необходимо выполнить команду git add (это многофункциональная команда, она используется для добавления под версионный контроль новых файлов, для индексации изменений, а также для других целей, например для указания файлов с исправленным конфликтом слияния). Выполним git add, чтобы проиндексировать benchmarks.rb, а затем снова выполним 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

#

Теперь оба файла проиндексированы и войдут в следующий коммит. В этот момент вы, предположим, вспомнили одно небольшое изменение, которое вы хотите сделать в benchmarks.rb до фиксации. Вы открываете файл, вносите и сохраняете необходимые изменения и вроде бы готовы к коммиту. Но давайте-ка ещё раз выполним 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

#

# Changes not staged for commit:

# (use "git add <file>..." to update what will be committed)

#

# modified: benchmarks.rb

#

Что за чёрт? Теперь benchmarks.rb отображается как проиндексированный и непроиндексированный одновременно. Как такое возможно? Такая ситуация наглядно демонстрирует, что Git индексирует файл в точности в том состоянии, в котором он находился, когда вы выполнили команду git add. Если вы выполните коммит сейчас, то файл benchmarks.rb попадёт в коммит в том состоянии, в котором он находился, когда вы последний раз выполняли команду git add, а не в том, в котором он находится в вашем рабочем каталоге в момент выполнения git commit. Если вы изменили файл после выполнения git add, вам придётся снова выполнить git add, чтобы проиндексировать последнюю версию файла:

$ 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

#

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

$ 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

Эта команда сравнивает содержимое вашего рабочего каталога с содержимым индекса. Результат показывает ещё не проиндексированные изменения.

Если вы хотите посмотреть, что вы проиндексировали и что войдёт в следующий коммит, вы можете выполнить git diff --cached. (В Git'е версии 1.6.1 и выше, вы также можете использовать git diff --staged, которая легче запоминается.) Эта команда сравнивает ваши индексированные изменения с последним коммитом:

$ git diff --cached

diff --git a/README b/README

new file mode 100644

index 0000000..03902a1

--- /dev/null

+++ b/README2

@@ -0,0 +1,5 @@

+grit

+ by Tom Preston-Werner, Chris Wanstrath

+ http://github.com/mojombo/grit

+

+Grit is a Ruby library for extracting information from a Git repository