

**Санкт-Петербургский национальный
исследовательский университет
информационных технологий, механики и
оптики**

Кафедра информатики и прикладной математики

Основы программной инженерии

Лабораторная работа №2

“Системы контроля версий “

Вариант: 212213

Выполнил: **Шкаруба Н.Е.**

Проверила: **Харитоновна А.Е.**

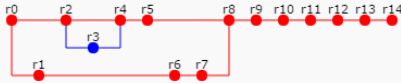
группа: **Р3218**

год: **2015**

Задание и блок-схема:

Вариант

212213



Сконфигурировать в своём домашнем каталоге репозитории svn и git и загрузить в них начальную ревизию файлов с исходными кодами (в соответствии с выданным вариантом).

Воспроизвести последовательность команд для систем контроля версий svn и git, осуществляющих операции над исходным кодом, приведённые на блок-схеме.

При составлении последовательности команд необходимо учитывать следующие условия:

- Цвет элементов схемы указывает на пользователя, совершившего действие (красный - первый, синий - второй).
- Цифры над узлами - номер ревизии. Ревизии создаются последовательно.
- Необходимо разрешать конфликты между версиями, если они возникают.

Полезная информация к защите

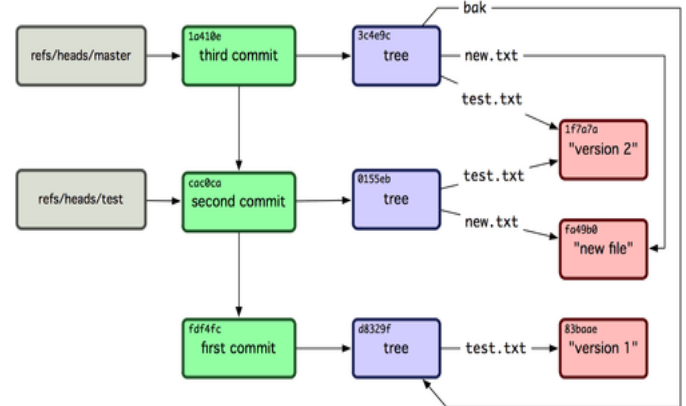
- По своей сути Git — контентно-адресуемая файловая система, т.е. простое хранилище ключ-значение.
- Каталог .git полностью представляет собой весь репозиторий (включая историю и метаданные). Включает в себя: HEAD, branches/, config, description, hooks/, index, info/, objects/, refs/.
- Git хранит данные сходным с файловыми системами UNIX способом, но в немного упрощённом виде. Содержимое хранится в объектах-деревьях, blob'ax (binary large object) и объектах-commit'ax. Объект-дерево может содержать одну и более записей, каждая из которых представляет собой набор из SHA-1 хеша, соответствующего блобу или поддереву, режима доступа к файлу, типа и имени файла.
- Каждый объект представляет из себя набор из SHA-1 хеша. К примеру, выведем последнее дерево master в нашем проекте:

```
$ git cat-file -p master^{tree}
100644 blob f1dc699d055b28221496    D.java
040000 tree 99f1a6d12cb4b6f19c8a    imaginedLib
```

- Схематично, данные в git'е хранятся так ----->
- Данные о коммитах хранятся в объектах-коммитах, формат которого прост: указатель на дерево верхнего уровня, имена автора, коммитера, временная метка, пустая строка и затем сообщение коммита.
- Когда выполняются команды { git add }, { git commit } - Гит сохраняет blobs для изменённых файлов, обновляет индекс, записывает объекты-деревья и коммит-объекты, ссылающиеся на деревья верхнего уровня и предшествующие коммиты.
- Git сжимает новые данные при помощи zlib, вычисляя для них SHA-1 хэш, являющийся названием.
- Ветка в Git'е — простой указатель или ссылка на последнюю версию в работе.
- Когда выполняется команда { git branch (brName) }, Git, по сути, просто добавляет хэш последнего коммита под указанным именем в виде новой ссылки.
- HEAD - символическая ссылка на текущую ветку. Отличается от обычной тем, что она не содержит сам хеш SHA-1, а содержит указатель на другую ссылку. Вскрытие head:

```
$ cat .git/HEAD
ref: refs/heads/master
```
- При выполнении git commit Git создаёт объект-коммит, указывая его родителем тот объект, SHA-1 которого содержится в файле, на который ссылается HEAD.
- Объект-метка очень похож на объект-коммит: он содержит имя поставившего метку, дату, сообщение и указатель. Разница в том, что ветка указывает на коммит, а не на дерево.
- Ссылки на удалённые ветки (ссылки в refs/remotes) отличаются от обычных веток (ссылки в refs/heads) тем, что на них нельзя переключиться с помощью git checkout. Git работает с ними как с закладками, указывающими на последнее состояние соответствующих веток на ваших серверах.
- Время от времени Git упаковывает несколько похожих объектов в один pack-файл. При упаковке Git ищет файлы, которые похожи по имени и размеру, и сохраняет только разницу между двумя версиями файла. Вторая версия сохраняется "как есть", а исходная — в виде дельты, т.к. доступ ко второй версии более вероятен.
- Можно вручную запускать Garbage collector для упаковки файлов и уплотнения ссылок путём { git gc -auto }, но, как правило, эта команда ничего не делает, т.к. Git делает это автоматически.
- Если накапливается слишком много ссылок, git их упаковывает в один файл с следующим содержанием:

```
$ cat .git/packed-refs
# pack-refs with: peeled
cac0cab538b970a37ea1e769cbbde608743bc96d refs/heads/experiment
ab1afef80fac8e34258ff41fc1b867c702daa24b refs/heads/master
```
- Если когда-либо в проект был добавлен большой файл, каждый, кто потом захочет клонировать проект, будет вынужден скачивать этот большой файл, даже если он был удалён в следующем же коммите. Он будет в базе всегда, просто потому, что он доступен в истории.



1. Git

Команды, используемые при выполнении задания:

Initialization:

- git help - git init - git config

Making changes:

- git add - git rm - git commit - git reset

Working with branches:

- git branch - git merge - git checkout - git diff

Synchronize changes:

- git clone - git push - git pull - git fetch

Листинг выполненных команд:

/* Перед каждым { git add . } , я изменял содержимое рабочего каталога */

```
r0: $ git help
    $ git config --global user.name 'Nikita Shkaruba'
    $ git config --global user.email Sh.nickita@list.ru
    $ git init
    $ git add .
    $ git commit -m 'r0'

r1: $ git branch dumbestIdea
    $ git checkout dumbestIdea
    $ git add .
    $ git commit -m 'r1'

r2: $ git checkout master
    $ git add .
    $ git commit -m 'r2'

r3: $ git config --global user.name 'Daniil Popov'
    $ git config --global user.email Phlash9@mail.ru
    $ git branch Danya-hotFix
    $ git checkout Danya-hotFix
    $ git add .
    $ git commit -m 'r3'

r4: $ git config --global user.name 'Nikita Shkaruba'
    $ git config --global user.email Sh.Nickita@list.ru
    $ git checkout master
    $ git merge --no-ff --no-commit Danya-hotFix
    $ git status
    $ git rm --cached UTPbMeg3f2.aXL
    $ git rm --cached
    $ git commit -m 'r4'
    $ git branch -d Danya-hotFix

r5: $ git add .
    $ git commit -m 'r5'

r6: $ git checkout dumbestIdea
    $ git add .
    $ git commit -m 'r6'

r7: $ git add .
    $ git commit -m 'r7'

r8: $ git checkout master
    $ git merge --no-ff --no-commit
    $ git status
    // solving conflict
    $ git add D.java
    $ git add .
    $ git commit -m 'r8'
    $ git branch -d dumbestIdea

r9: $ git add .
    $ git commit -m 'r9'

r10: $ git add .
    $ git commit -m 'r10'

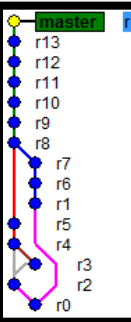
r11: $ git add .
    $ git commit -m 'r11'

r12: $ git add .
    $ git commit -m 'r12'

r13: $ git add .
    $ git commit -m 'r13'

r14: $ git add .
    $ git commit -m 'r14'
```

Результат:



master	r14	Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 03:29:48
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 03:28:56
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 03:28:26
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 03:27:41
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 03:27:18
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 03:26:53
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 03:21:26
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 03:14:18
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 03:12:57
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 02:41:59
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 03:10:44
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 03:04:07
		Daniil Popov <Phlash9@mail.ru>	2015-10-11 02:46:52
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 02:44:08
		Nikita Shkaruba <Sh.Nickita@list.ru>	2015-10-11 02:38:54

2. Svn (Subversion)

Команды, используемые при выполнении задания:

Initialization:

- svn help - svnadmin create - svn checkout

Making changes:

- svn add - svn delete - svn commit - svn revert

Working with branches:

- svn copy - svn merge -svn switch

Листинг выполненных команд:

```
r0:
$ svn help
$ mkdir svn-repository
$ mkdir workflow
$ cd Svn-repository
$ svnadmin create -fs-type fsfs .
$ cd ../workflow
$ svn checkout file:///C:/Svn-repository
$ svn add * // first commit includes a trunk/ , branches/ , tags/ folders.
$ svn commit --username=Nikita -m "r0"

r1:
$ svn copy file:///C:/Svn-repository/trunk file:///C:/Svn-repository/branches/dumbestIdea -m "Create branch
dumbestIdea"
$ svn switch file:///C:/Svn-repository/branches/dumbestIdea
$ svn add * --force
$ svn commit --username=Nikita -m "r1"

r2:
$ svn switch file:///C:/Svn-repository/trunk/
$ svn add * --force
$ svn commit --username=Nikita -m "r2"

r3:
$ svn switch file:///C:/Svn-repository/branches/Danya-hotFix
$ svn add * --force
$ svn commit --username=Danya -m "r3"

r4:
$ svn switch file:///C:/Svn-repository/trunk
$ svn merge file:///C:/Svn-repository/branches/Danya-hotFix
$ svn add * --force // resolving conflicts
$ svn commit --username=Nikita -m "r4"

r5:
$ svn add * --force
$ svn commit --username=Nikita -m "r5"

r6:
$ svn switch file:///C:/Svn-repository/branches/dumbestIdea
$ svn add * --force
$ svn commit --username=Nikita -m "r6"

r7:
$ svn add * --force
$ svn commit --username=Nikita -m "r7"

r8:
$ svn switch file:///C:/Svn-repository/trunk/
$ svn merge file:///C:/Svn-repository/branches/dumbestIdea
$ svn add * --force // resolving conflicts
$ svn commit --username=Nikita -m "r8"

r9:
$ svn add * --force
$ svn commit --username=Nikita -m "r9"

r10:
$ svn add * --force
$ svn commit --username=Nikita -m "r10"

r11:
$ svn add * --force
$ svn commit --username=Nikita -m "r11"

r12:
$ svn add * --force
$ svn commit --username=Nikita -m "r12"

r13:
$ svn add * --force
$ svn commit --username=Nikita -m "r13"

r14:
$ svn add * --force
$ svn commit --username=Nikita -m "r14"
```

Результат:

Message	Revision	Branch	Author	Date
trunk r14	17	trunk	Nikita	01:36 AM
r13	16	trunk	Nikita	01:34 AM
r12	15	trunk	Nikita	01:33 AM
r11	14	trunk	Nikita	01:31 AM
r10	13	trunk	Nikita	01:30 AM
r9	12	trunk	Nikita	01:29 AM
r8	11	trunk	Nikita	01:27 AM
dumbestIdea r7	10	dum...	Nikita	01:13 AM
r6	9	dum...	Nikita	01:12 AM
r5	8	trunk	Nikita	01:10 AM
r4	7	trunk	Nikita	01:03 AM
Danya-hotFix r3	6	Dany...	Danya	12:48 AM
Created whole branch just for me	5	Dany...	Nikita	12:44 AM
r2	4	trunk	Nikita	12:41 AM
Create branch dumbestIdea	3	dum...	Nikita	12:35 AM
r1	2	dum...	Nikita	12:16 AM
r0	1	trunk	Nikita	12:08 AM