Краткое руководство по использованию системы контроля версий GIT

Система контроля версий GIT предназначена для контролируемого внесения изменений в процессе *разработки* программного обеспечения, а также при написании статей и другом роде деятельности, связанном с редактированием текстовых файлов (для удобства в дальнейшем будем обсуждать только тексты программ). Система позволяет *отследить происходящие изменения* в исходных текстах программ, что делает удобной совместную разработку.

Механизм *ветвей* позволяет с легкостью проводить эксперименты над проектом, не затрагивая основного кода, либо вести раздельную разработку модулей, которые впоследствии могут быть включены в основной проект выполнением операции *слияния*.

Внесенные изменения можно контролировать с помощью протокола, который ведет система. Кроме того, система контроля версий в автоматическом режиме отслеживает корректность вносимых изменений при одновременной разработке и предотвращает ситуации противоречащих друг другу исправлений в коде.

Легкая в использовании и одновременно богатая функционально, система контроля версий Git призвана стать незаменимым инструментом любого разработчика!

Возможности Git:

1. Контроль версий программ
   - автоматическое *резервное копирование* текущей версии программы и соответствующих исходных кодов;
   - удобная *сводка* внесенных изменений;
   - интеллектуальное отслеживание *противоречащих* друг другу изменений;
2. Совместная разработка программы группой программистов:
   - контроль ответственности: *кто, когда и какие изменения* внес в исходный код;
   - *статистика* вклада участников в разработку проекта;
3. Создание версий разработки
   - проведение *экспериментов*, не затрагивая основного кода программы;
   - совместная *модульная разработка:* отдельные модули, которые в дальнейшем включаются в общий проект;
   - легкое *управление* версиями: слияние версий в основной проект, перемещение версий, удаление версий;
4. Использование системы контроля версий для написания статей
   - *совместное* написание с контролем ответственности и разделением обязанностей;
   - удобство отслеживания *изменений*;

Начальная конфигурация

Для первоначальной настройки системы контроля версий достаточно указать имя разработчика и электронный адрес. Сделать это можно из командной строки, предварительно осуществив вход на кластер по ssh, с помощью команд

```
user@krc-master:~> git config --global user.name "Ваше Имя"
user@krc-master:~> git config --global user.email your@e-ma.il
```

Создание нового проекта

Для того, чтобы система GIT начала отслеживать изменения в исходных файлах, достаточно

в директории с текстами программ ввести команды

```
user@krc-master:~>git init
user@krc-master:~>git add .
user@krc-master:~>git commit -a -m"First version"
```

Первая команда в этой последовательности создает пустой репозиторий в подпапке .git/ текущей папки, вторая команда добавляет текущую версию всех имеющихся файлов в этой папке в качестве отслеживаемых, третья осуществляет запись «снимка» текущей версии в репозитории.

Внесение изменений

После редактирования файлов можно вручную просмотреть, какие именно файлы были изменены. Для этого используется команда

```
user@krc-master:~>git status
```

Эта команда выводит на экран список всех измененных файлов. Если все верно, можно зафиксировать изменения при помощи команды git commit с опцией -a (т. е. Автоматически добавить все измененные файлы в репозиторий)

```
user@krc-master:~>git commit -a -m "Added new function to main.cpp"
```

Отметим полезную опцию -m, которая позволяет добавлять краткое описание внесенных изменений, что в дальнейшем позволяет локализовать ошибки и упрощает разработку.

Следует заметить, что команда commit -a не добавляет новых файлов к репозиторию. Если пользователь создает новый файл в текущем проекте и желает включить его в отслеживание, это следует сделать вручную командой add:

```
user@krc-master:~>git add file1.txt file2.cpp file3.tex
```

Просмотр истории изменений

Для этого служит команда

```
user@krc-master:~>git log
```

Кроме того, у этой команды есть подробный режим пошагового просмотра изменений

```
user@krc-master:~>git log -p
```

Также можно посмотреть всю историю разработки при помощи ключей

```
user@krc-master:~>git log --stat --summary
```

Ветви

Механизм ветвей позволяет проводить эксперименты, не нарушая основной линии развития программы. Редактирование отдельной ветви программы не затрагивает исходного кода ветви **master**, поэтому безопасно. Для создании ветви следует воспользоваться командой

```
user@krc-master:~> git branch experimental
```

После этого можно проверить, сколько в данный момент ветвей в программе при помощи команды

```
user@krc-master:~> git branch
  experimental
* master
```

При этом текущая «рабочая» ветвь помечена звездочкой. Для смены ветви необходимо воспользоваться командой git checkout:

```
user@krc-master:~> git checkout experimental
```

После завершения редактирования можно сохранить изменения и вернуться на основную ветвь

```
user@krc-master:~> git commit -a -m "Added some experiments"
user@krc-master:~> git checkout master
```

Для внесения изменений из экспериментальной ветви в основную ветвь следует воспользоваться командой git merge

```
user@krc-master:~> git merge experimental
```

Для удаления ветви (с проверкой внесения изменений в основную ветвь) используется команда

```
user@krc-master:~> git branch -d experimental
```

Удаление ветви без внесения изменений в основную ветвь делается командой

```
user@krc-master:~> git branch -D experimental
```

Копия проекта на локальной машине

Предположим, что пользователю требуется редактировать исходные файлы одновременно на кластере и на локальной машине. Будем считать, что «основная» версия проекта находится на кластере.

Если проект состоит из одной ветви **master**, то именно эта ветвь будет активна, поэтому ее удаленное изменение по умолчанию будет запрещено. Для того, чтобы разрешить изменения с других источников «основной» версии, необходимо, находясь на кластере, выполнить команду конфигурирования

```
user@krc-master:~/my_project>git config receive.denyCurrentBranch ignore
```

Затем с «локальной» машины выполняем команду копирования репозитория:

```
user@local:~/my_version>git clone user@82.196.66.9:~/my_project ./
```

В результате в текущей папке (в примере папка my_version в домашнем каталоге пользователя на локальной машине) будет создана копия проекта my_project с кластера. После этого можно производить изменения на локальной машине, завершить которые необходимо командой git commit:

```
user@local:~/my_version>git commit -a -m "Added some changes"
```

После этого содержимое «локальной» версии будет отличаться от «основной». Их слияние необходимо обеспечить выполнением команды git push

```
user@local:~/my_version>git push origin
```

В результате исходные файлы на кластере и на локальной машине будут одинаковыми. Однако, прежде чем приступать к редактированию «основной» версии проекта, находясь на кластере, необходимо выполнить команду (в противном случае возникнут серьезные ошибки)

```
user@krc-master:~>git checkout -f
```

В дальнейшем перед началом редактирования «локального» проекта необходимо загрузить свежую версию «основного» проекта командой

```
user@local:~/my_version>git pull origin
```