**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ**

**ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«Севастопольский государственный университет»**

**Институт информационных технологий и управления   
в технических системах**

**РАБОТА С ВЕТКАМИ в GIT**

**Методические указания**

к лабораторной работе №3

по дисциплине

**“Управление IT-проектами”**

для студентов специальности 09.03.02 –

"Информационные системы и технологии"

всех форм обучения

**Севастополь**

**2015**

1. **Цель работы:** изучить особенности ветвления в Git.
2. **ОСНОВНЫЕ ПОЛОЖЕНИЯ**
   1. **Алгоритм работы над задачей**

Стандартный алгоритм работы над какой-либо задачей выглядит так:

1. Создаётся ветка, основывающаяся на последней копии master ветки. Название новой ветки содержит класс задачи, краткое описание, номер задачи. Например, feature/sessions\_add\_datetime\_filter.5503

2. Все изменения производятся внутри этой ветки. При каждом атомарном логическом изменении (например, добавили плагин – закоммитили добавление; поправили API одной функции во всех местах – закоммитили и т.п.) создаётся свой коммит. Все это позволяет разделять предыдущие изменения, упрощается чтение и проверка на ошибки процесса разработки.

3. После того, как код в ветке сделан и отлажен, все изменения закоммичены, данная ветка ребейзится относительно последнего мастера, и пушится в центральный репозиторий.

4. Второй человек, работающий над тем же проектом, делает к себе pull центрального репозитория. Если он уже смотрел – то удаляет свою локальную копию ветки, после чего переключается на указанную ветку. Прочитывает код, проверяет его работоспособность, после чего либо отдаёт на доработку, если там обнаружены проблемы, либо делает еще раз rebase поверх мастера, и слияние ветки с мастером.

5. После слияния с мастером, ревьюер пушит новый мастер в центральный репозиторий, удаляет у себя локальную ветку задачи, пушит в мастер удаление ветки задачи.

6. Разработчик удаляет локальную ветку задачи после того, как задача была закрыта и изменения попали в master.

**Несколько рекомендаций по работе с Git:**

1. Команда git log –oneline – показывает созданные коммиты
2. Чтобы перейти в другую папку воспользуйся командой “cd”. Не используются обратные слеши «\», а только обычные «/». Чтобы открыть содержимое диска D:\ напишите cd /D.
3. Команда cd .. – переход на уровень вверх
4. Для просмотра содержимого текущей папки воспользуйтесь командой ls.
5. Находясь под Windows можно не обращать внимание на заглавные/строчные буквы в названиях файлов и папок.
6. Вместо ctrl + c – используется ctrl + insert

Вместо ctrl + v – используется shift + insert

ctrl + c – используется для остановки текущего, запущенного в консоли процесса.

* 1. **Ветвление в Git. Основные команды**

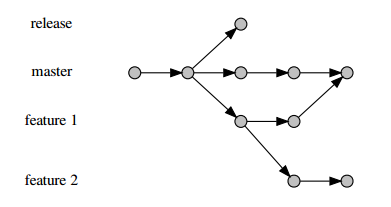
Наличие веток – одна из наиболее важных особенностей Git. Работу с репозиторием Git можно представить как цепочку коммитов, а ветка – параллельная основной (и другим веткам) цепочка коммитов, берущая своё начало в одном из уже существующих коммитов. Ветвление означает, что вы отклоняетесь от основной линии разработки и продолжаете работу, не вмешиваясь в основную линию.

Основной веткой Git, как правило, считается ветка **master**, но Git на самом деле устроен так, что все ветки равнозначны. Ветка “master” в Git – это не специальная ветка. Она точно такая же, как и все остальные ветки. Она существует почти во всех репозиториях только лишь потому, что ее создает команда git init, а большинство людей не меняют ее название.

Редактирование отдельной ветви программы не затрагивает исходного кода ветви **master**. Умение обращаться с ветками в Git требуется любому разработчику, планирующему работать в команде.

Cоздание и удаление веток совершенно не затратно, так как ветка в Git – это всего лишь файл, содержащий 40 символов контрольной суммы SHA-1 того коммита, на который он указывает. Создание новой ветки совершенно быстро и просто – это всего лишь запись 41 байта в файл (40 знаков и перевод строки). Это совершенно отличает Git от ветвления в большинстве более старых систем контроля версий, где все файлы проекта копируются в другой подкаталог. Там ветвление для проектов разного размера может занять от секунд до минут. В Git ветвление всегда мгновенное.

Рассмотри пример.



Имеется четыре ветки: *master* (основная ветка по умолчанию), *feature 1*, *feature 2*, *release*.

Ветка master – основная ветка, на которую ориентируются все разработчики. В данной ветке аккумулируются все нововведения, из неё делают финальную версию программы.

На рисунке изображена ветка release, функция данной ветки – зафиксировать исходный код программы, который был на момент выхода стабильной версии (релиза). Ветка master при этом отодвинулась от release на несколько шагов.

Ветка feature 1 – некоторая возможность программы, которая развивается отдельно от хода разработки в ветке master. Спустя некоторое время ветка feature 1 была объединена с master, о чём свидетельствует стрелка из feature 1 в master, это означает, что код, разработанный в рамках ветки feature 1, теперь находится внутри ветки master.

Ветка feature 2 – ещё одна параллельно развиваемая возможность программы, которая основана на ветке feature 1. Здесь нет стрелки, переходящей в master, это говорит о том, что ветка ещё не была включена в основной исходный код программы.

Git не накладывает ограничений на использование веток, поэтому их можно использовать в различных смыслах этого слова. Рассмотрим эти возможности.

**1.** Ветки можно использовать аналогично системам управления версиями предыдущего поколения: каждая ветка – это какая-то новая возможность в программе, экспериментальная версия.

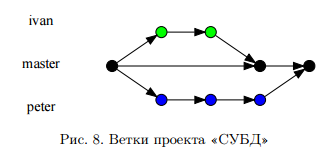
**2.** Простота работы с ветками в Git позволяет создавать для каждого разработчика свою собственную ветку, и эта возможность очень часто практикуется в Git-репозиториях: каждый разработчик создаёт для себя собственную ветку и работает только в ней. Разработчики не мешают друг другу (при этом никто не препятствует объединению любой «чужой ветки» со «своей»), а роль менеджера в данном случае – объединять ветки с master, когда разработчик выполнил свою работу.

**3.** Ветки можно использовать просто для отдельных проектов; например, многие проекты с открытым исходным кодом используют ветки, чтобы размещать там исходный код своего сайта или документации.

**4.** Управление версиями может быть также организовано при помощи веток. В Git нет понятия «версия», но версии можно организовать в Git сразу несколькими способами.

Для более наглядного представления рассмотрим пример. В организации (назовём её «Х») работают два разработчика (назовём их Иван и Пётр) и один менеджер проекта. Данная организация создаёт собственную систему управления базами данных – «СУБД».

Первый год команда работает над первой версией своего продукта, при этом Иван работает в своей ветке ivan, а Пётр, соответственно, peter. Спустя полгода работы наступает время тестирования, ветки ivan и peter отправляются на рассмотрение менеджеру и объединяются с веткой master. Следующие полгода программа тестируется, в ней исправляются ошибки и периодически происходит объединение веток с master. Спустя год разработки дерево коммитов программы выглядит, как показано на рис.

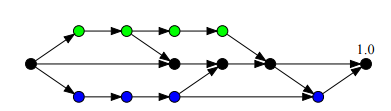
****

По окончании тестирования все ветки завершаются и последнему коммиту в ветке master присваивается версия 1.0.

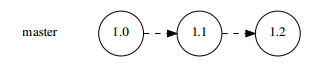
Для этого выполняется команда:

$ git tag "1.0"

Получившееся дерево коммитов показано на рис.

****

После выпуска версии 1.0 у организации «Х» появляются первые клиенты, которые работают с базой данных версии 1.0. Спустя ещё несколько лет выпускаются версии 1.1 и 1.2, с которыми также работают клиенты.

****

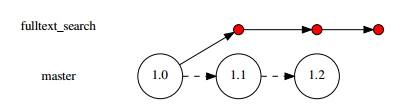
Предположим, что одному из клиентов, работающему на версии 1.0, потребовалась реализация полнотекстового поиска в базе данных. Он решил, что ему не нужно обновление до версии 1.1 или 1.2, так как ни в одной из версий требуемой возможности нет. Вместо этого клиент просит организацию «Х» реализовать ему данную возможность для версии «СУБД» 1.0, чтобы обновить систему с минимальными усилиями. Git позволяет очень легко это сделать в несколько шагов.

1. Перейти к версии 1.0 (git checkout "1.0").

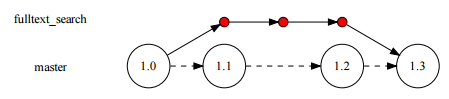
2. Создать новую ветку с именем «fulltext\_search», что будет означать «полнотекстовый поиск» (для этого нужно выполнить команду git branch fulltext\_search).

3. Приступить к разработке.

4. Полученную ветку скомпилировать и отдать заказчику. После выполнения вышеперечисленных шагов дерево коммитов примет следующий вид, как показано на рис.

****

Разработанная возможность может пригодиться другим клиентам, поэтому вполне логичным шагом будет объединить данную ветку со следующей версией. Поэтому с выходом версии 1.3 менеджер проекта объединяет ветку fulltext\_search с master.

****

Для создания ветви следует воспользоваться командой

$ git branch experimental

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

$ git branch

Результат

experimental

\* master

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

$ git checkout experimental

Чтобы создать ветку и сразу переключиться на нее, можно выполнить команду git checkout с параметром -b:

**$** git checkout -b experimental

Это тоже самое что и:

**$** git branch experimental

**$** git checkout experimental

После завершения редактирования можно сохранить изменения и вернуться на основную ветвь

$ git status

$ git add .

$ git commit -m "Added some experiments"

$ git checkout master

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

$ git merge experimental

Ещё одна полезная возможность для выяснения состояния веток состоит в том, чтобы оставить в этом списке только те ветки, которые вы слили (или не слили) в ветку, на которой сейчас находитесь. Для этих целей в Git’е есть опции --merged и --no-merged.

Чтобы посмотреть те ветки, которые вы уже слили с текущей, можете выполнить команду git branch --merged:

**$** git branch --merged

experimental

\* master

Чтобы увидеть все ветки, содержащие наработки, которые вы пока ещё не слили в текущую ветку, выполните команду git branch --nomerged:

**$** git branch --no-merged

testing

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

$ git branch -d experimental

Удаление ветви без внесения изменений в основную ветвь делается командой

$ git branch -D experimental

1. **ПРАКТИЧЕСКАЯ ЧАСТЬ**
2. Создайте ветку
3. Проверьте изменения (команда status)
4. Добавьте новый файл

*Выполните:*

touch <имя файла2. >

1. Зафиксируйте изменения

Теперь в Вашем проекте есть две ветки.

*Выполните:*

git hist --all или git log

Добавление опции --graph в git log вызывает построение дерева коммитов с помощью простых ASCII символов.

1. Выполните переключения к ветке master.
2. Сделайте изменения в ветке master.
3. Закомментируйте текущие изменения.
4. Вернитесь к созданной ранее ветке.
5. Просмотрите отличие текущих веток

git hist --all

Метка --all гарантированно означает, что видны все ветки. По умолчанию показывается только текущая ветка.

1. Объедините ветки.
2. Вернитесь к созданной ветке в точку перед слиянием с master (необходимо найти последний коммит перед слиянием).
3. Проверьте ветку.
4. Поищите лог новой ветки.
5. Вернитесь в master и создайте конфликт.
6. Разрешите конфликт.
7. Сделайте коммит.
8. Объедините ветки.
9. **СОДЕРЖИМОЕ ОТЧЁТА**

Отчёт должен содержать в себе:

1. Титульный лист.

2. Краткое описание работы с ветками Git.

3. Практическая работа над выбранным ранее проектом с использованием команд ветвления Git.

3. Вывод о проделанной работе (полученные навыки, замеченные трудности или, наоборот, улучшения в работе).

4. Список использованных источников

1. **КОНТРОЛЬНЫЕ ВОПРОСЫ**
2. Для чего предназначена система управления версиями исходного кода программы?
3. Что такое репозиторий?
4. Из чего формируется история изменений в репозитории?
5. Для чего используется механизм ветвления?
6. Что такое система Git? Какова её область применения?
7. В чём заключается ключевое отличие системы Git от систем управления версиями предыдущего поколения?
8. В чём различие централизованных и распределённых систем управления версиями?
9. Смогут ли разработчики продолжить работу с репозиторием (просмотр истории изменений, сравнение версий, создание новых коммитов) после выхода из строя центрального хранилища исходных текстов в случае централизованной системы управления версиями? В случае распределённой? Ответ поясните.
10. Требуется ли в системе Git для выполнения коммитов и просмотра истории изменений доступ в сеть?
11. Что представляет собой интерфейс системы Git?
12. Как получить справку по командам системы Git?
13. Какие первичные настройки нужно выполнить перед началом работы с Git? Нужно ли дублировать первичную настройку на другом компьютере?
14. Можно ли создать репозиторий Git в уже существующем проекте с исходным кодом?
15. Если в рабочем каталоге создать новый репозиторий, в каком состоянии будут находиться индекс и репозиторий?
16. В каком случае нужно выполнить коммит сразу после того, как был создан новый репозиторий? Ответ поясните.
17. Как добавить в репозиторий файл из рабочего каталога?
18. Как избежать случайного добавления ненужных файлов в репозиторий?
19. Как добавить в репозиторий несколько файлов? Все файлы рабочего каталога?
20. Возможно ли выполнить команду git commit, не указывая комментарий (параметр -m)? Что произойдёт в данном случае?
21. Был создан новый каталог, в него положили новый файл и инициализировали репозиторий. Какие действия необходимо выполнить перед коммитом, чтобы этот файл оказался в репозитории?
22. В Git есть команда, которая проверяет состояние репозитория и выводит соответствующие подсказки – git status. Приведите как минимум два примера, в каких ситуациях данная команда может быть полезна.
23. Какую полезную информацию о репозитории можно получить с помощью команды git status?
24. Приведите примеры использования веток в Git.
25. Перечислите основные команды ветвления в Git.
26. Перечислите основные возможности при использовании веток в Git.
27. **РЕКОМЕНДУЕМЫЕ ИНТЕРНЕТ-ИСТОЧНИКИ**
28. Git How То – интерактивный тур по Git на русском языке: <http://githowto.com/ru>.
29. Интерактивный учебник Git: <http://tty.github.com/>.
30. Интерактивный учебник по ветвлению Git, работающий в браузере: <http://pcottle.github.io/learnGitBranching/>.
31. Сайт github.com является хорошим справочником по Git. Справочная система GitHub находится по адресу: <http://help.github.com/>.
32. Интерактивный учебник "Pro Git" <http://progit.org/book/>
33. Интерактивный учебник "Git Community Book" <http://book.git-scm.com/>
34. Книга "Git Magic" <http://www-cs-students.stanford.edu/~blynn/gitmagic/index.html>,

перевод: <http://habrahabr.ru/blogs/Git/80909/>

1. Сравнение команд Git-SVN <http://git.or.cz/course/svn.html>
2. Редактирование истории в git <http://gq.net.ru/2009/12/16/git-history-rewrite/>