Лабораторная работа № 2

Системы контроля версий

Цель работы

1. Изучить на практике понятия и компоненты систем контроля версий (СКВ), порядок и приемы работы с ними.
2. Освоить специализированное ПО и распространенные сервисы для работы с распределенной СКВ Git.

Задание на лабораторную работу

1. Отработать навыки использования хранилища на локальной машине.

1.1. Настроить Git, указав имя и e-mail разработчика для подписи commit-ов.

Команда в терминале (и аналогичная для `user.email`):

```
git config --global user.name 'Dmitry Kozlyuk'
```

Примечание. Настройки Git хранятся для всей системы, для пользователя (файл в домашнем каталоге) и для проекта (файл `.git/config` в репозитарии). Нужно настроить пользовательские, global в терминах Git. E-mail можно задавать любой — Git просто включает его в коммиты, но больше никак не использует.

1.2. Создать новый проект Code::Blocks, функцию `main()` сделать пустой.

1.3. Инициализировать хранилище Git в каталоге созданного проекта.

Команда в терминале (из каталога проекта): `git init`

1.4. Выбрать файлы для отслеживания и совершить коммит.

1.4.1. Просмотреть состояние рабочей копии (команда `git status`). Пояснить в отчете, что означает каждая строка в выводе команды.

1.4.2. Добавить файл в будущий коммит: `git add main.cpp`

1.4.3. Повторить пункт 1.4.1 и пояснить отличия в отчете.

1.4.4. Выполнить коммит командой `git commit`.

Указание. О написании сообщения к коммиту см. в указаниях к ЛР.

1.5. Создать новые коммиты разными способами и в разных условиях.

Указание. Здесь и далее при выполнении однотипных пунктов полезно работать сначала в терминале, затем в GUI, чтобы выделить суть независимо от программы.

1.5.1. Создать второй коммит с файлом проекта (`*.cbp`).

1.5.2. Добавить в программу ввод двух целых чисел с приглашением.

1.5.3. Повторить пункт 1.4.1 и пояснить отличия в отчете.

1.5.4. Добавить измененный файл в индекс и совершить третий коммит.

1.5.5. Добавить в программу вывод суммы введенных чисел и совершить четвертый коммит.

Указание. Командой `git add -u` можно добавить в индекс все файлы, которые отслеживались Git и были изменены, а командой `git commit -a` можно сразу добавить их в индекс и выполнить коммит.

1.6. Предотвратить добавление в хранилище файлов, не нуждающихся в контроле версий, — `*.obj`, `*.exe`, `*.layout` и `*.depend`.

1.6.1. В файле `.gitignore` в корне репозитария записать шаблоны игнорирования: `*.obj` и прочие, по правилу на строку. Можно игнорировать и каталоги с ненужными файлами, например, `/bin`.

Указание. Создать этот файл в Windows можно так: `touch .gitignore`

1.6.2. Сделать коммит, содержащий `.gitignore`.

1.6.3. Просмотреть состояние рабочей копии, чтобы удостовериться, что игнорирование действует так, как требуется.

1.7. Просмотреть историю (журнал) хранилища командой `git log`.

Указание. Полезно сравнить вид истории в GUI и в терминале командой

```
git log --graph --pretty=oneline --abbrev-commit
```

1.8. Просмотреть содержимое коммитов командой `git show` *`ref`*, где *`ref`*:

— `HEAD`: последний коммит;

— `HEAD~1`: предпоследний коммит (и т. д.);

— `b34a0e`: коммит с указанным хэшем.

1.9. Просмотреть разность между двумя несмежными пунктами истории командой `git diff` *`ref1 ref2`*.

1.10. Освоить возможность отката к заданной версии.

1.10.1. Удалить весь код из `main.cpp` и сохранить файл.

1.10.2. Удалить все несохраненные изменения в файле командой:

```
git checkout -- main.cpp
```

1.10.3. Повторить пункт 1.10.1 и сделать коммит.

1.10.4. Откатить состояние хранилища к предыдущей версии командой

```
git reset --hard HEAD~1
```

2. **Освоить передачу истории хранилища по сети.**

2.1. Организовать хранилище на удаленном сервере.

2.1.1. Зарегистрироваться на GitHub или на BitBucket с именем пользователя формата `KozlyukDA` (если такой вариант категорически неприемлем, можно выбрать иной псевдоним и сообщить преподавателям).

2.1.2. Создать пустое удаленное хранилище под названием `sdt-lab01`.

Указание. Вопреки инструкции на GitHub, добавлять в хранилище файл `README.md` не нужно, удаленный репозитарий должен быть пустым.

2.2. Настроить локальное хранилище для синхронизации с удаленным.

В терминале достаточно команды `git remote add origin` *URL*, где URL — адрес репозитария, указанный на сайте, где создано хранилище. Имя `origin` является типовым для единственного удаленного репозитария. Командой `git remote -v` можно отобразить все удаленные хранилища. В TortoiseGit хранилище добавляется в диалоге *Settings*, пункт *Remote*.

Указание. Доступ может осуществляться по разным протоколам, в частности, по HTTP, которым и следует воспользоваться.

2.3. Передать историю на удаленный сервер командой `git push`.

Указание. Изначально Git не считает ветки в локальном и удаленном репозитарии связанными, поэтому будет предложена модифицированная команда `git push`, которую и следует выполнить. Далее `git push` не будет требовать аргументов.

Указание. Здесь и далее при взаимодействии с удаленным сервером потребуется вводить имя пользователя и пароль, с которыми выполнялась регистрация на GitHub или BitBucket. TortoiseGit может сохранять этот пароль, что мешает, когда ПК используется разными людьми. В разделе *Credentials* настроек можно удалить сохраненные данные.

2.4. Перейти к странице хранилища на GitHub или BitBucket (обновить её) и ознакомиться с возможностями просмотра содержимого репозитария через web-интерфейс.

2.5. Загрузить копию удаленного хранилища на локальную машину командой `git clone` *`URL каталог`*. 

Указание. Целью является имитация совместной работы с удаленным хранилищем. Для этого на одной машине организуются 2 локальных хранилища: созданное в пункте 1.2 (каталог `RepoA`) и загруженное с удаленного сервера (каталог `RepoB`).

3. Освоить способы параллельной работы над проектом без конфликтов кода.

3.1. Создать параллельные изменения.

3.1.1. В локальном хранилище `RepoB` добавить в программу печать разности введенных чисел, сделать коммит и передать изменения на сервер.

3.1.2. В локальном хранилище `RepoA` добавить над функцией `main()` комментарий о том, что программа является учебной, сделать коммит, **но не отправлять изменений на сервер.**

3.2. На странице хранилища на GitHub перейти в раздел *Commits* и ознакомиться с возможностью просмотра истории изменений через web-интерфейс.

3.3. В локальном хранилище `RepoA` выполнить загрузку с сервера новейших ветвлений и изменений командой `git fetch` и просмотреть журнал хранилища. 

Указание. По умолчанию показывается только текущая активная ветвь (в данном случае — `master`). Просмотреть все commit-ы во всех ветвях, в том числе в загруженной из удаленного хранилища ветви `origin/master`, нужно включить флажок «All branches» слева снизу окна журнала. В терминале нужен ключ `--all`.

3.4. Перебазировать локальные изменения в `RepoA` на загруженные с сервера командой `git rebase origin/master`  . Отправить результат на сервер.

3.5. В `RepoB` загрузить изменения с сервера. Командой `git pull --ff-only` переместить ветку master к новейшему коммиту, полученному с сервера. 

4. Изучить действия, связанные с ветвлениями и разрешением конфликтов.

Указание.. На данном этапе во всех трех хранилищах (локальных `RepoA` и `RepoB` и удаленном на GitHub) должна быть одинаковая линейная история из пяти — шести commit-ов. Все операции выполняются в одном локальном хранилище, например, в `RepoA`.

4.1. Добавить в программу печать произведения чисел и совершите commit.

На данном этапе программа может быть такой:

```
1  #include "sdt.h"
2
3  // This program is just an example one under VCS.
4  int main()
5  {
6      int a, b;
7      cout << "Enter A and B: ";
8      cin  >> a >> b;
9      cout << "A + B = " << a + b << '\n'
10          << "A - B = " << a - b << '\n'
11          << "A * B = " << a * b << '\n';
12 }
```

4.2. Создать новую ветвь под названием `division`. из пункта истории, в котором был добавлен комментарий над `main()`, и перейти на нее. 

В терминале выполнить команду: `git checkout -b division HEAD~1`

Это эквивалентно двум командам — созданию ветви и переходу на нее:

```
git branch division HEAD~1
git checkout division
```

Указание. Переключаться можно только при чистом (clean) хранилище, то есть, без изменений в рабочей копии. В TortoiseGit переключиться на ветвь можно, выбрав в контекстном меню коммита, которым эта ветвь оканчивается, пункт «Switch/checkout to this». При создании ветви можно сразу установить флажок «Switch to new branch».

4.3. В новой ветви повторить пункт 0, заменив умножение делением.

4.4. Переключиться обратно на ветвь `master`.

4.5. Выполнить слияние ветви `division` в ветвь `master` так, чтобы в последней оказался код для печати и произведения, и частного, командой `git merge division`  . Результатом ожидается конфликт.

Указание. TortoiseGit отображает конфликт как окно с сообщением об ошибке — но не следует закрывать его!

4.6. Ознакомиться с тем, как представляется в Git конфликт при слиянии.

4.6.1. Просмотреть состояние рабочей копии, зафиксировать его в отчете и пояснить каждый элемент (строку).

4.6.2. Просмотреть в любом редакторе файл, содержащий конфликт. Найти и зафиксировать в отчете область конфликта (фрагмент файла).

4.6.3. При использовании TrotoiseGit просмотреть конфликт в графическом виде, нажав кнопку *Resolve* (снизу), а затем выбрав пункт *Edit conflicts* из контекстного меню `main.cpp`. 

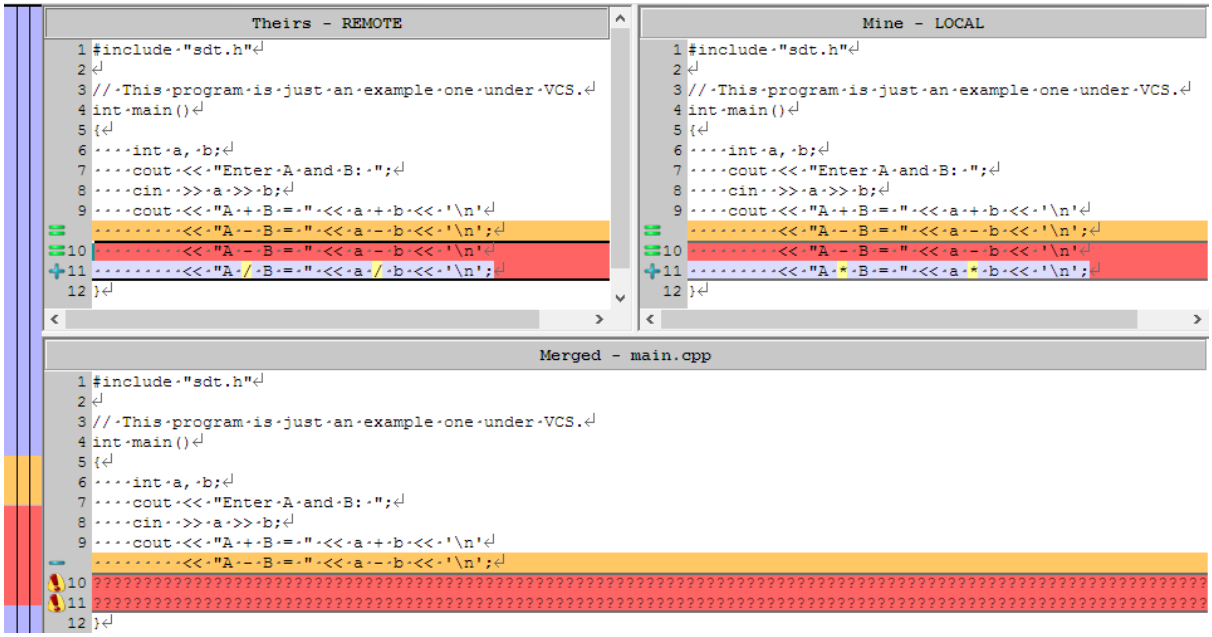

Рисунок 1 — Вид участка конфликта в TortoiseGitMerge

4.7. Разрешить конфликт. Для этого нужно в любом редакторе — в Code::Blocks, Vim или TortoiseGitMerge — исправить место конфликта на такой код, который должен получиться в результате слияния. Метки участка конфликта следует убрать.

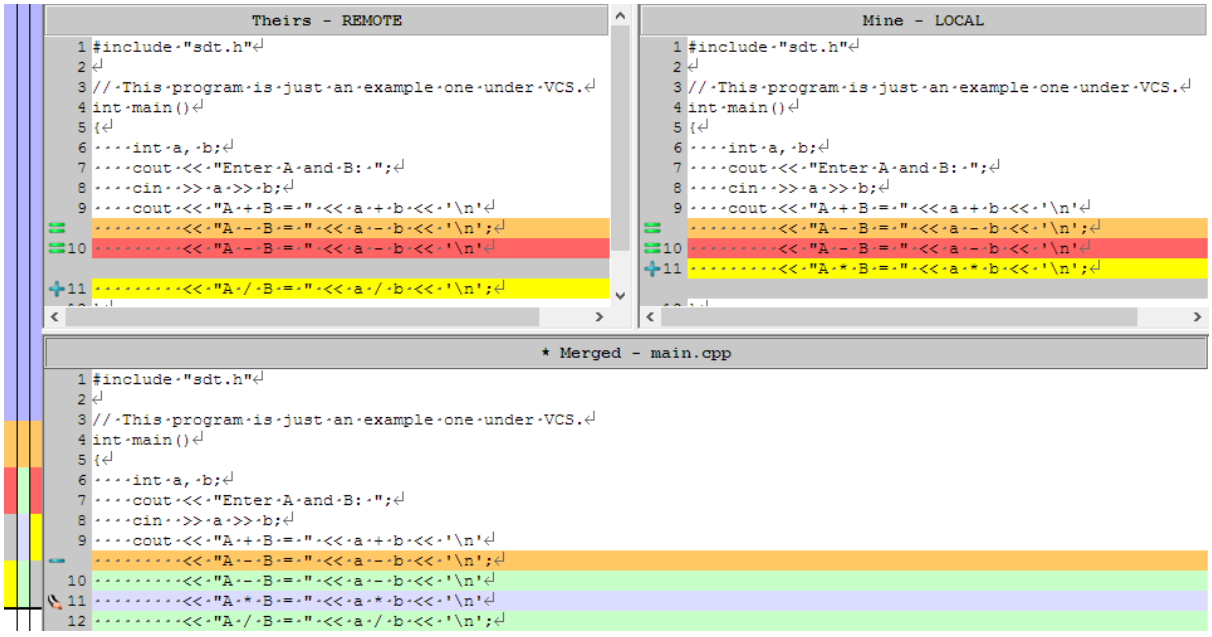

Рисунок 2 — Возможный способ разрешения конфликта

4.8. Завершить процедуру слияния.

В терминале нужно добавить файл с разрешенным конфликтом в индекс и сделать коммит. В TortoiseGitMerge нужно нажать кнопку «Mark as resolved», чтобы отметить файл как избавленный от конфликтов, и закрыть программу, а затем сделать коммит.

4.9. Убедиться, что программа компилируется и верно работает. Если это не так, исправить все ошибки и добиться правильной работы.

Указание. Ситуация, когда после слияния программа все-таки оказывается не вполне корректной, случается на практике довольно часто. В этом случае коммит, созданный при слиянии, оказывается логически неправильным, он не имеет ценности без последующего исправления. В Git имеется возможность изменить (amend) уже совершенный коммит, пока он не передан на сервер. Это делается при коммите с исправлением: следует установить флажок «Amend Last Commit» в диалоге создания коммита — нового не появится, а вместо этого изменения будут приписаны предыдущему пункту истории. В терминале это флаг `--amend` у команды `git commit`. Можно воспользоваться данной возможностью при выполнении пункта.

4.10. Передать все изменения всех ветвей в удаленное хранилище.

Указания к выполнению лабораторной работы

Задания необходимо читать внимательно и полностью. Благодаря тому, что Git является распределенной СКВ, резервную копию локального хранилища можно сделать простым копированием или архивацией.

В отчет можно включать снимки экрана, сообщения и комментарии по своему усмотрению с тем, чтобы было удобно пояснять сделанное, опираясь на отчет. Если пункт выполнялся и из терминала, и через GUI, достаточно включить в отчет текстовый вариант.

Выбор программного обеспечения

Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним в *nix можно получить из терминала, а в Window — из специальной консольной оболочки Git Bash, запускаемой из контекстного меню любой папки. Часто используются и графические программы, например, TortoiseGit (Windows), GitG (Linux), SourceTree (Windows и OS X) и другие. Третий вариант — средства работы с Git из IDE (Visual Studio, IDEA, Eclipse).

Терминальный режим удобен полным контролем происходящего и поэтому рекомендуется при освоении, в частности, для ЛР. Значки  в тексте — ссылки на руководство по Git. Графические средства удобнее для просмотра истории и изменений. На ЛР предлагается пользоваться TortoiseGit — расширением «Проводника» Windows. Значки  в тексте — ссылки на руководство по TortoiseGit.

В настоящее время (2016 г.) широко применяются web-интерфейсы к репозитариям, доступным по сети. Из программ наиболее распространены cgit, GitLab, Stash и Gogs. Однако, еще более популярны сайты-сервисы — GitHub и BitBucket.

Написание сообщений к коммитам

Составление текста

На практике важно, чтобы примечание к коммиту было информативным: в будущем по заголовкам читают быстро историю проекта, а также ищут коммиты по ключевым словам. Заголовок (первая строка) должен быть коротким и описывать суть изменений, потому что он показывается в списке коммитов. Часто в заголовок включают тему (к какой части проекта относится коммит) или ссылки на номер задачи в bugtracker-е. Например:

`build: используется новая версия CMake` — коммит относится к сборке

`обрабатывает пустой массив | fix #1234` — исправляет ошибку № 1234

`timer: учет високосных лет #4321`— доработка таймера по задаче № 4321

Из заголовка должно быть ясно, что в целом сделано и зачем (почему, для чего).

После заголовка через пустую строку может идти расширенное пояснение. Случается, что по размеру сообщение больше изменений в исходном коде и включает гиперссылки, примеры, описания исправленных ошибок.

Использование редактора

Git Bash для написания сообщения к коммиту предлагает редактор Vim. Чтобы начать вводить текст в нем, нужно нажать `i` (одну клавишу). Чтобы закончить ввод — Escape, чтобы сохранить текст и выйти, нужно закончить ввод и дважды нажать `Z` (Shift+Z).