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.

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 (2017 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.

Git Bash

1. Git Bash запускается из контекстного меню любой папки пунктом «Git Bash Here».
2. Переход в заданную директорию *path* осуществляется с помощью команды `cd` *path*. При этом путь к папке может быть, как относительный (`cd folder1`), так и абсолютным (`cd /c/Users/имя/Desktop/folder1/`). Переход в директорию выше уровнем осуществляется командой `cd ..` (две точки в конце).
3. Печать полного пути до текущей директории осуществляется командой `pwd`.
4. Печать имён файлов в текущей директории осуществляется командой `ls` или же в заданной — `ls <директория>`. Для отображения скрытых файлов (имя которых начинается с точки) необходимо использовать `ls -A`.
5. Вставка текста в консоль Git Bash осуществляется сочетанием клавиш Shift+Insert, скопировать выделенный в консоли мышкой текст можно сочетанием Ctrl+Insert. Также работает вставка средней кнопкой мыши (щелчок колесиком).
6. Можно перемещаться по истории введённых команд при помощи клавиш «вверх» и «вниз», работает дополнение имен клавишей Tab.

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.

Внимание! Если комментарий к коммиту пуст, Git не будет его выполнять.

Использование редактора

Git Bash для написания сообщения к коммиту предлагает редактор Vim (он не является частью Git). Альтернатива для коротких сообщений:

```
git commit -m 'текст сообщения'
```

Чтобы начать вводить текст в Vim, нужно нажать `i` (одну клавишу), чтобы закончить ввод — Escape. Чтобы сохранить текст и выйти, нужно закончить ввод (Escape), ввести `:x` (отобразится внизу) и нажать Enter. **Внимание!** Нажатие Ctrl+Z вместо Shift+Z оставит редактор включенным, а коммит незавершенным; нужно вернуться в Vim командой `fg`.

Строки, начинающиеся с октоторпа (`#`) в сообщения не войдут.

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:

   ```
   git config --global user.name 'Дмитрий Козлюк'
   git config --global user.email 'KozliukDA@mpei.ru'
   ```

   Примечание. Адрес может быть вымышленным, Git не использует его.

   1.2. Создать на рабочем столе папку `labs/lab02/RepoA`.

   1.3. Создать новый проект в CLion с кодом по умолчанию в данной папке.

   1.4. Инициализировать хранилище Git в каталоге созданного проекта.

   Команда в терминале (здесь и далее — из каталога проекта):

   ```
   git init
   ```

   1.5. Выбрать файлы для отслеживания и совершить коммит.

   1.5.1. Просмотреть состояние рабочей копии (команда `git status`).

   Пояснить в отчете, что означает каждая строка в выводе команды.

   1.5.2. Добавить файл в будущий коммит:

   ```
   git add main.cpp
   ```

   1.5.3. Повторить пункт 1.5.1 и пояснить отличия в отчете.

   1.5.4. Выполнить коммит командой `git commit`.

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

1.6. 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 уяснить суть независимо от программы.

1.6.1. Создать второй коммит с файлом `CMakeLists.txt`.

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

На данном этапе программа должна быть такой:

```
#include <iostream>

using namespace std;

int main()
{
    int a, b;
    cout << "Enter A and B: ";
    cin  >> a >> b;
}
```

Указание. Здесь и далее текст программы должен соответствовать заданию: он составлен так, чтобы было удобно изучать различные аспекты управления версиями в том порядке, как предлагается в ЛР.

1.6.3. Повторить пункт 1.5.1 и пояснить отличия в отчете.

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

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

Указание. В конец `main()` добавить строку:

```
cout << "A + B = " << a + b << '\n';
```

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.

1.7. Предотвратить добавление в хранилище файлов и каталогов, которые не нуждаются в контроле версий, — `.idea/`, `cmake-build-debug/`. 1.7.

В файле `.gitignore` в корне репозитария (нужно создать его)

перечислить игнорируемое, по правилу на строку.

Указание. Создать этот файл в Git Bash можно так: `touch .gitignore`. Редактировать можно «Блокнотом» или дописыванием строк командой:

```
echo .idea/ >> .gitignore
```

1.7.2. Сделать коммит, содержащий `.gitignore`.

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

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

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

```
git log --graph --oneline --decorate
```

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

- `HEAD`: последний коммит;
- `HEAD~1`: предпоследний коммит (и т. д.);
- `b34a0e`: коммит с указанным хэшем.

1.10. Просмотреть разность между двумя несмежными пунктами истории командой `git diff` *`REF1 REF2`*.

1.11. Освоить возможность отката к заданной версии.

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

1.11.2. Удалить все несохраненные изменения в файле командой:

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

1.11.3. Повторить пункт 1.11.1 и сделать коммит.

1.11.4. 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`.

Указание. Электронная почта должна быть действующей.

2.1.2. Создать пустое удаленное хранилище под названием `sdt-lab02`.

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 (**внимание:** не web-страницы — см. рис. 1). Имя `origin` является типовым для единственного удаленного репозитария. Командой `git remote -v` можно отобразить все удаленные хранилища.

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

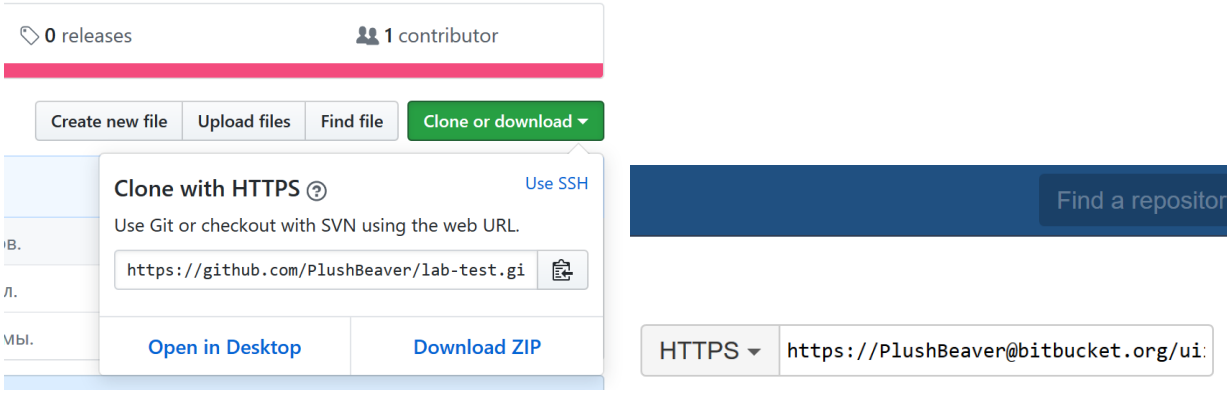

Рисунок 1 — Адреса хранилищ в интерфейсе GitHub (слева) и BitBucket (справа)

2.3. Передать историю на удаленный сервер :

```
git push --set-upstream origin master
```

Здесь и далее при взаимодействии с удаленным сервером потребуется вводить имя пользователя и пароль, с которыми выполнялась регистрация на GitHub или BitBucket.

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

Указание. В отчет нужно внести снимок экрана, на нужно котором отметить ссылки и кнопки, позволяющие просматривать коммиты и файлы.

2.5. Загрузить копию удаленного хранилища на локальную машину командой `git clone` *`URL каталог`*, где *`каталог`* — `labs/lab02/RepoB`.

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 синхронизацию последовательных доработок проекта

3.1. В локальном хранилище `RepoA` добавить в программу печать первого из чисел, умноженного на 2, сделать коммит и передать изменения на сервер командой `git push`.

3.2. Выполнить `git pull` в локальном хранилище `RepoB`, чтобы загрузить последние изменения с сервера. Просмотреть историю и убедиться, что был получен последний коммит из `RepoA`.

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

4. **Освоить способы параллельной работы над проектом без конфликтов кода.**

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

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

4.3. 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`.

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

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

5. **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. Все операции выполняются в одном локальном хранилище, например, в `RepoA`.

5.1. Добавить в программу печать произведения чисел и совершить коммит.

На данном этапе программа должна быть такой:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int a, b;
8      cout << "Enter A and B: ";
9      cin  >> a >> b;
10     cout << "A + B = " << a + b << '\n'
11          << "A * 2 = " << a * 2 << '\n'
12          << "A - B = " << a - b << '\n'
13          << "A * B = " << a * b << '\n';
14 }
```

5.2. 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».

5.3. В новой ветви повторить пункт 5.1, заменив умножение делением.

5.4. Переключиться обратно на ветвь `master`.

5.5. 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!

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

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

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

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

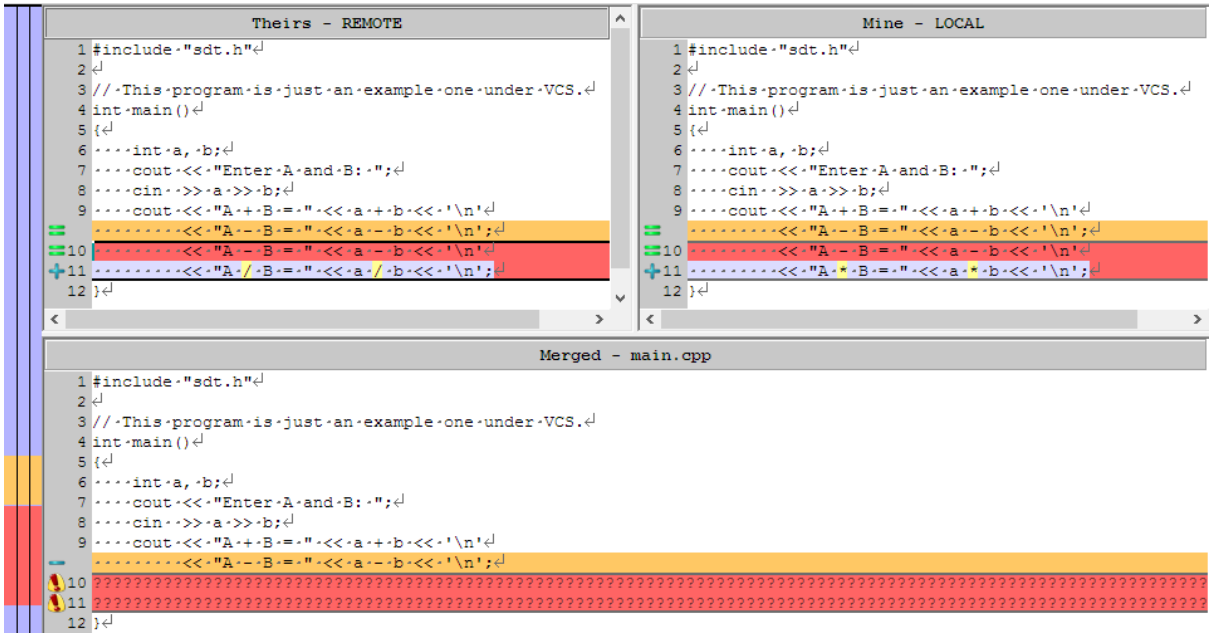

Рисунок 2 — Вид участка конфликта в TortoiseGitMerge

5.7. Разрешить конфликт. Для этого нужно в любом редакторе исправить место конфликта на такой код, который должен получиться в результате слияния. Метки участка конфликта следует убрать.

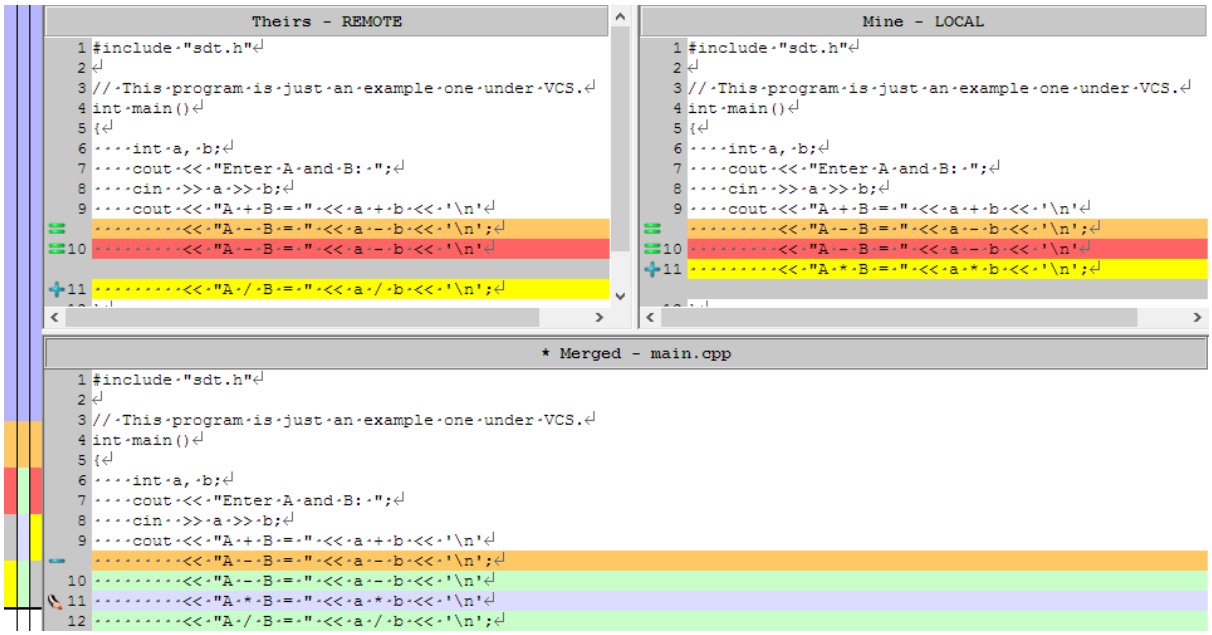

Рисунок 3 — Возможный способ разрешения конфликта

5.8. 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.

5.9. 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.

5.10. Передать все изменения всех ветвей в удаленное хранилище.