Лабораторная работа

Система управления версиями Git

Подготовил А.В.Бруханский

Распределенная система управления версиями Git

Система управления версиями (СУВ) Git позволяет контролировать изменения файлов в выбранных папках на своем компьютере и согласовывать эти изменения с изменениями файлов на компьютерах членов команды, совместно работающих над каким-либо проектом. Возможно согласование изменений с хранилищем файлов проекта (репозиторием) на выделенном сервере, в том числе на серверах общедоступных сервисов https://github.com и https://bitbucket.org.

После загрузки программного обеспечения Git с сервера https://git-scm.com и его установки на компьютере в списке установленных программ появляется папка Git с ссылками на приложения Git Bash, Git CMD и Git GUI. Git Bash – интерпретатор командной строки *bash* ОС UNIX/Linux перенесенный в Windows и интегрированный с Git. Интеграция с Git позволяет интерпретатору выполнить команды Git из любой текущей папки и хранить все настройки программы для использования при последующих вызовах. Git CMD – интерпретатор *cmd* командной строки Windows, также понимающий команды Git. Для выполнения команд Git в окне Git CMD необходимо командой *cd* сделать текущей папкой папку с контролируемыми файлами.

Git GUI – графический интерфейс пользователя программы Git. С его помощью можно выполнить основные операции Git по управлению файлами проекта в локальном и удаленном (remote) репозиториях без необходимости знания синтаксиса команд Git. Однако в некоторых случаях его возможностей оказывается недостаточно для более сложных случаев совместной работы над проектами.

После установки Git на компьютере в контекстном меню правой кнопки мыши появляются команды запуска *Git Bash Here* и *Git GUI Here*, позволяющие запустить интерпретатор команд и графическую оболочку с привязкой к открытой папке с контролируемыми файлами проекта или папке, которую нужно сделать рабочей папкой проекта.

Настройка Git

1. Откройте папку, содержащую файлы проекта одной из ранее выполненных лабораторных работ. Из контекстного меню правой кнопки мыши выберите *Git Bash Here* и в открывшемся окне введите команду:

   $ *git init*

В результате в папке с проектом появится папка с именем «.git» , содержащая все необходимые файлы локального репозитория.

2. Авторизуйтесь для внесения последующих изменений, указав свое имя и адрес электронной почты, например :

   $ *git config user.name "Максим Лунин"*

   $ *git config user.email max.lunin@mail.ru*

   Если вы собираетесь постоянно работать с проектами на данном компьютере, то добавьте к командам авторизации ключ *global*:

   $ *git config --global user.name "Сергей Васин"*

   $ *git config --global user.email serge_vasin@yandex.ru*

3. Создайте в папке с проектом файл «.gitignore» для постоянного исключения из числа файлов с отслеживаемыми изменениями временные и автоматически создаваемые файлы компиляторов и компоновщиков. При создании файла с именем «.gitignore» средствами контекстного меню мыши возможен отказ системы Windows сохранить файл без указания имени перед точкой. В этом случае сохраните его с произвольным именем, например, «a.gitignore», а затем в командной строке cmd или Git CMD переименуйте его командой ren, предварительно сделав папку с файлом текущей:

   >*ren a.gitignore .gitignore*

Переименование файла можно выполнить также с помощью файлового менеджера типа «Total Commander».

Создайте в файле «.gitignore» список файлов и папок, отслеживать которые не нужно, например:

*~

*obj

Win32/

Debug/

__history/

Символ «*» в указании имени заменяет произвольное количество символов, а символ «/» после имени обозначает папку.

4. Добавление имен удаленных репозиториев.

   При совместной работе над проектом нескольких разработчиков вначале необходимо получить файлы текущей версии проекта из удаленного репозитория. Для этого необходимо знать краткое имя сервера, на котором расположен удаленный репозиторий, или его URL-адрес. Знания краткого имени будет достаточно, если файлы в рабочую папку уже копировались с сервера командой *git clone*, например

   $ *git clone https://github.com/имя_владельца/имя_папки_проекта* .

   При этом по умолчанию удаленному серверу присваивается имя «*origin*».

   Проверить присвоенные имена серверов можно командой *git remote* :

   $ *git remote –v* .

   Ключ *–v* позволяет вывести полные URL-адреса серверов:

   origin https://github.com/имя_владельца/имя_папки (*fetch*)

   origin https://github.com/имя_владельца/имя_папки (*push*)

   В скобках указаны команды скачивания (*fetch*) и отправки изменений (*push*) допустимые для этого сервера.

   Добавить еще одно имя сервера с удаленным репозиторием можно командой *git remote add* :

   $ *git remote add краткое_имя_сервера URL-репозитория* .

   Добавление имени сервера имеет смыл только после регистрации своего аккаунта на соответствующем сайте или после получения *URL*-адреса от руководителя разработки проекта.

Основные команды Git

5. Проверка состояния рабочей папки осуществляется командой

$ *git status* .

В ответ Git Bash покажет название текущей ветви «master», присвоенное по умолчанию, и список всех неотслеживаемых файлов в рабочей папке, не подпадающих под шаблоны файла «.gitignore». Имена этих файлов будут выделены красным шрифтом.

6. Включение файлов и папок в перечень отслеживаемых выполняется командой git add :

   $ *git add имена_файлов_или_шаблоны_имен* .

   Например, команда

   $ *git add *cpp *dfm *h* .

   включает в перечень отслеживаемых файлы исходных текстов на языке C++ (*cpp), описание графической формы (*dfm) и заголовочные файлы (*h).

7. Внесите изменения в один из текстовых файлов, например, добавив комментарий к какому-либо оператору исходного текста программы на языке C++, и сохраните измененный файл. Снова выполните команду *git status*. Перечень отслеживаемых файлов рабочей папки, не затронутых изменениями и подготовленных к фиксации их состояния командой *commit* будет выведен зеленым шрифтом. Имя измененного файла и файлов, не включенных в отслеживаемые, отобразится красным шрифтом.

8. Добавьте измененный файл к файлам, подготовленным к фиксации состояния (staged files), с помощью команды

   $ *git add имя_файла* .

   Проверьте состояние файлов командой *git status* и сохраните снимок текущего состояния отслеживаемых файлов командой *git commit* :

   $ *git commit -m 'Ver 1.0'* .

   Ключ «*-m*» позволяет сохранить сообщение (message) *'Ver 1.0'* как комментарий, описывающий данную фиксацию (снимок состояния). Сообщение должно быть простым текстом на любом языке, заключенным в апострофы или двойные кавычки. В данном примере предлагается

указать, что этот снимок состояния является первой версией файлов проекта. Любая фиксация требует обязательного указания строки сообщения.

Ввод команды *git commit* без указания сообщения запускает текстовый редактор, которым по умолчанию является стандартный редактор текста ОС UNIX/Linux *Vim*. Редактор *Vim* всегда запускается в *командном режиме*. Чтобы перевести его в *режим вставки*, то есть в режим ввода и изменения текста, необходимо ввести с клавиатуры символы «*i*» или «*a*». В первой пустой строке открывшего текста-комментария нужно ввести сообщение, описывающее данную фиксацию, а затем перейти в командный режим нажатием клавиши <Esc>. Выход из редактора с сохранением изменений выполняется командой

:wq или ***ZZ*** .

При этом будет программа git выполнит фиксацию состояния файлов проекта с сохранением в качестве комментария строки введенного сообщения.

9. Просмотрите историю изменений с помощью команды *git log*:

$ *git log* или $ *git log --oneline* .

Ключ «--*oneline*» существенно сокращает объем выводимой информации. Ключ «-p» позволяет вывести различие между каждым коммитом, а ключ «-*число*» – ограничивает вывод количеством коммитов, заданных *числом*. Например, команда

$ *git log -p -3*

выведет информацию о трех последних коммитах с указанием различий в отслеживаемых файлах.

10. Создание новой ветки

Необходимость создания новой ветки в разработке проекта возникает, если предполагаются существенные отклонения от первоначального замысла или разработчикам требуется одобрение результатов работы руководителем проекта.

6

Создайте ветку *testing* командой

$ *git branch testing* .

Узнать, какая ветка является текущей можно, введя команду *git log*:

$ *git log* или $ *git log --oneline* .

В первой строке будет выведена информация о наличии двух веток

1bdf852 (HEAD -> master, testing) текст_сообщения_commit,

на которые указывает указатель *HEAD.*

Переход на ветку на новую выполняется командой *git checkout*.

$ git checkout testing .

Повторный ввод команды *git log --oneline* покажет, что переход произошел:

1bdf852 (HEAD -> testing) текст_сообщения_commit.

11. Внесите изменение в один-два отслеживаемых текстовых файла новой ветки и зафиксируйте изменения командой

$ git commit -a -m "текст_комментария_commit"

Ключ «*-a*» в команде *git commit* позволяет пропустить команду *git add,* добавляющую измененные файлы в область индексирования, подготавливая их к фиксации.

Просмотрите историю изменений ветки *testing* командой *git --oneline* .

12. Слияние веток.

Перейдите на главную ветку *master* командой

$ git checkout master .

Выполните слияние ветки *testing* с веткой *master* вводом команды

$ git merge testing

Просмотрите историю изменений командой

$ git log --graph --oneline

Ключ «*--graph*» позволяет графически показать историю разделения и слияния веток *master* и *testing*:

** ae208a3 (HEAD -> master) Merge branch 'testing'*

|\

| * *44a04e1 (testing) Исправлен Text1.txt*

| * *5033ec5 Исправлен Text2.txt*

|/

* *5b12bf1 (origin/master) Добавлен Text2.txt*

* *4789148 Добавлен Test1.txt*

Посмотрите результат объединения исправленных файлов в текстовом редактере (блокноте).