Часть 1 (Git)

Предисловие

Git – распределенная система контроля версий (Distributed Version Control System).
Системы контроля версий бывают локальные (их репозитории хранятся на локальной машине), централизованные (их репозитории хранятся централизованно на сервере) и распределенные (их репозитории хранятся на узлах сети).
При использовании Git для восстановления данных (в общем случае) достаточно, чтобы они были хотя бы на одной машине — концепция распределенной системы (управления версиями) может быть задействована достаточно глубоко. При работе на локальной машине работа с данными происходит в первую очередь на этой машине, а синхронизация с распределенным репозиторием происходит в последствии (и при наличии возможности в этом). При отсутствии возможности синхронизации она выполняется при появлении возможности в этом — работу с данными это не ограничивает (фиксирование изменений все равно возможно).
Git хранит данные в виде снимков их состояний. Во избежание дублирования идентичных снимков вместо идентичных снимков сохраняются просто ссылки на ранее сохраненные снимки.
При сохранении изменений вычисляется контрольная сумма (SHA- 1) изменяемых файлов — и факт изменения безусловно становится известен Git. Поиск файлов осуществляется системой контроля версий по этой контрольной сумме.
С точки зрения Git, файлы (с которыми в Git работает пользователь) могут находится в следующих состояниях: committed (изменения сохранены пользователем), modified (изменения внесены пользователем), staged (изменения помечены пользователем для дальнейшего сохранения).
В Git работа с данными ведется в следующих папках: рабочая папка (это пользовательская папка, в нее пользователь выгружает данные из репозитория для работы с ними), область staged (это служебная папка, в ней хранятся файлы в состоянии staged), папка Git (это служебная папка, в ней хранится локальный репозиторий, в целях экономии он хранится сжатым).
Классический процесс работы в Git выглядит так: файлы в рабочей папке модифицируются в ходе работы с ними, помечаются пользователем как staged, и затем изменения сохраняются пользователем.

О названии

Git наиболее точно/актуально переводится на русский как «слабое звено».

Название было выбрано именно таким, видимо, согласно причине появления этого средства. `Git` была разработана в интересах дальнейшей разработки OS Linux – разработчики нуждались в специфической системе контроля версий (контроль версий был слабым звеном этого процесса).
`Git` была разработана в 2K6, и в течение дальнейших нескольких лет стала повсеместно применяема.

В завершение предисловия

Дальнейший материал посвящен в основном командам `Git` – команды будут указаны через приветствие утилиты командной строки (через `$`).

Прочие сведения

Прочие сведения касательно настройки утилиты Git – см. другие источники.

Часть 2 (Инсталляция)

Инсталляция и настройка

Инсталляция

Инсталляция утилиты Git выполняется средствами мастера установки. Мастер установки — это утилита с GUI, и ее использование не требут каких- либо прочих комментариев.

Настройка

Выполняется инструментом `git config`. Настройка во многом сводится к установке переменных настройки. Эти переменные хранятся так:
файл `etc/gitconfig` содержит переменные настройки всех пользователей, и всех репозиториев; для доступа к этим переменным утилита `git config` запускается с ключем `--system`;
файл `~/.config/git/config` содержит переменные настройки конкретного пользователя; для доступа к этим переменным утилита `git config` запускается с ключем `--global`;
файл `.git/config` в конкретном репозитории содержит настройки этого конкретного репозитория; для доступа к этим переменным утилита `git config` запускается с ключем `--global`;

эти файлы указаны в соответствии с приоритетом хранящихся в них настроек (чем дальше по списку тем выше приоритет).
Места хранения настроек следует уточнять для конкретной ОС.

Идентификатор пользователя

пользователю следует указать идентификатор, и адрес электронной почты — эти свведения будут включаться в коммиты;
это делается следующими командами:

```
$ git config --global user.name " … "
$ git config --global user.email …
```

Текстовый редактор

текстовый редактор задается ключем:

```
$ git config --global core.editor …
```

текстовый редактор целесообразно оставить по умолчанию.
Подробнее о прочих настройках — см. другие источники.

Проверка настроек

список всех настроек возвращается ключем:

```
$ git config --list
```

конкретная настройка возвращается ключем:

```
$ git config …
```

Часть 3 (Использование)

Вызов справки

Справка по команде может быть вызвана одной из следующих команд:

```
$ git help …
$ git …  --help
$ man git- …
```

где … - это имя конкретной команды.

Создание репозитория

Репозиторий может быть создан с нуля (тогда каталог, в котором хранятся целевые данные, импортируется в `Git`), либо с готового репозитория (просто клонированием этого репозитория).
Чтобы создать репозиторий с нуля, импортировав целевой каталог, необходимо перейти в него, и ввести ключ:

```
$ git init
```

в результате в целевом каталоге возникнет папка `.git` – она и будет папкой локального репозитория (необходимо помнить, что непосредственно после этого действия создание репозитория будет лишь инициировано, папка `.git` будет пуста, и файлы рабочей папки еще не будут отслеживаемыми).
Чтобы файлы рабочей папки отслеживались `Git`, их нужно указать — это делается соответствующим ключем:

```
$ git add …
```

файлы можно указать по одному, или же не по одному — используя специфический синтаксис (`*`, `FileName.*`, `*.FileExtension`, `.`, etc. sintax).
Кроме того, это действие необходимо подтвердить:

```
$ git commit -m
```

Клонирование существующего репозитория выполняется командой:

```
$ git clone …
```

этой команде передается расположение репозитория (в общем случае это URL).
Клонирование репозитория выполняется над всем указанным репозиторием, и в результате будут склонированы все хранящиеся в нем версии (разумеется, при клонировании репозиторий пополняется склонированными данными, и при выгрузке в рабочую папку эти файлы будут отслеживаемыми).

В результате этой команды будет создана папка, одноименная папке клонируемого репозитория. Она будет содержать непосредственно в себе текущую версию репозитория, и папку `.git`, содержащую весь репозиторий.
Можно склонировать репозиторий и не в одноименную папку, таким ключем:
`$ git clone … FolderName`

и тогда он будет склонирован в папку `FolderName`.
Будучи создан, репозиторий готов к работе; при работе с репозиторием необходимо помнить, что интересующие пользователя файлы должны быть отслеживаемыми (команда `$ git add …` была рассмотрена выше).
В прочем, все файлы клонированного репозитория и так являются отслеживаемыми файлами.

Проверка состояния файлов

Можно проверить состояние файлов в рабочей папке на предмет modified/staged. Это делается ключем `$ git status`. Эта команда выдаст расположение по ветви (подробности — далее), состояние отслеживаемых файлов, и список не отслеживаемых файлов.
Команда `$ git status` по умолчанию выдает развернутый вывод.
Чтобы использовать краткий вывод, она запускается как `$ git status --short`, или `$ git status -s`.
В кратком выводе используется соответствующая система обозначений статусов. Обозначение статусов следующее:
`??` - не отслеживаемый файл, остальные файлы — отслеживаемые (обозначение отслеживаемых файлов состоит из двух символов — первый обозначает, изменялся ли файл перед staging, второй обозначает, изменялся ли файл после staging (это обозначение может отсутствовать);
все эти обозначения задаются соответствующими литерами — факт модификации обозначается литерой m);

Подробнее об обозначениях — см. другие источники.
Есть альтернативная команда — команда `$ git diff`. Эта команда выводит список изменений в рабочем каталоге. В выводе этой команды используются специфические обозначения. Эта команда имеет специфические ключи. Например, следующие ключи:
`--staged`, выведет список всех изменений, которые были staged;
без ключей, выведет список всех изменений, которые не были staged;

Добавление файлов для отслеживания

Файлы рабочей папки, за которыми не было отслеживания, можно сделать отслеживаемыми — командой `$ git add FileName`.
Необходимо помнить, что `Git` управляет версиями только отслеживаемых файлов.

Staging файлов

Чтобы пометить файлы для фиксации изменений, эти файлы должны быть отслеживаемыми. Файлы помечаются как staged той же командой `$ git add` (запись имеет вид `$ git add FileName`).
После того, как файлы были staged, они по прежнему могут быть отредактированы — тогда, для того чтобы измененные версии были staged, необходимо использовать эту команду еще раз, после изменений. Иначе же, будут staged те версии файлов, которые были staged в последний раз.
Эта команда способна учитывать все изменения, которые произведены над отслеживаемыми файлами рабочего каталога (в том числе, и удаление таких файлов; если удаление, произведенное пользователем (средствами файловой системы, не утилиты Git) будет staged посредством этой команды, то это изменение тоже будет отслеживаемым и совершенно корректным с точки зрения утилиты Git).

Игнорирование файлов

Игнорирование файлов позволяет игнорировать не интересующие пользователя файлы (в «статусе» они не будут отображаться даже как неотслеживаемые). Эту возможность следует использовать хотя бы из- за наличия системных файлов, которые и так не интересны пользователю.
Список игнорируемых файлов задается ключем:

```
$ cat .gitignore
 …
```

здесь … – это список игнорируемых файлов. Список файлов можно задать в конкретных файлах, или же обобщенно — с помощью спец. обозначений. Могут использоваться рассмотренные выше обобщенные обозначения, и обозначения пути к файлам. Подробнее обэтих обозначениях — см другие источники.

Фиксация изменений

Фиксация изменений выполняется командой `$ git commit`.
По этой команде вызывается текстовый редактор по умолчанию — это предусмотрено для составления сообщения о commit (при этом строки, стартующие с `#`, игнорируются, а пустое сообщение приведет к отмене commit). При запуске редактора в нем выводятся данные, аналогичные возвращаемым командой `$ git status` (эти данные выводятся закомментированными через символ `#`). Если же пользователь желает видеть в редакторе более подробные данные, то команду `$ git commit` следует вызывать с ключем `-v` (эти данные также выводятся закомментированными через `#`).
Эту команду можно использовать и проще:

```
$ git commit -m " … "
```

здесь " … " это текст сообщения; в данном случае текст сообщения задается и так, и поэтому текстовый редактор не запускается.
Необходимо помнить — чтобы включить в сообщение о commit строки, предлагаемые редактором по умолчанию, необходимо удалить символ #, с которого они стартуют.
При использовании команды `$ git commit` необходимо помнить, что фиксируются только те сведения, которые staged.
По факту исполнения этой команды в репозиторий добавляется целевой снимок.
Как и говорилось вше, чтобы зафиксировать изменения, они непременно должны быть staged. Однако, они могут быть staged как явно (вышеописанным способом), так и неявно — самой командой `$ git commit` (запись с ключем `$ git commit -a`), по вызову этой команды все (отслеживаемые) файлы будут (неявно) staged, и затем committed.

Удаление файлов

При необходимости файлы могут быть удалены из числа отслеживаемых. Это производится командой `$ rm …` , здесь … это имя файла. Эта команда приводит к удалению указанного файла из числа отслеживаемых файлов, и из рабочего каталога. Это изменение требует фиксации (фиксация производится повторным вызовом этой команды (команды `$ git rm`)). По сути, эта фиксация — специфический staging (касающийся изменений, произведенных именно данной операцией), и не отменяет необходимости настоящей фиксации командой `$ git commit`.
Если необходимо выполнить эту операцию над файлом, который уже был staged, необходимо использовать ключ `-f` (в данном случае, f это от force).
Если же необходимо просто отменить staging, то эта команда вызывается с соответствующим ключем (`$ rm … --cached`).

Перемещение/переименование

Команда перемещения/переименования файлов:
```
$ git mv OldFileName NewFileName
```

Перемещение/переименование должно выполняться этой командой, иначе же оно не будет отслеживаемым (если это изменение не будет зафиксировано). Разница между перемещением и переименованием файла этой командой заключается просто в том, что в случае перемещения указывается не только имя файла, а еще новый путь к перемещаемому файлу, по которому он и будет располагаться.

Лог версий

Лог версий вызывается командой:

```
$ git log
```

по вызову этой команды будет возвращен лог версий текущего репозитория. Лог выводится в таком порядке — первыми выводятся свежайшие коммиты.
Ключ `-2` этой команды ограничивает ее вывод двумя пунктами этого списка (на месте двойки можно указать любое число).
Ключ `-p` делает вывод более прецезионным (по сути, используя для этого вывод команды `$ git diff`).
Ключ `--stat` напротив сделает вывод команды более усеченным.
Ключ `--oneline` усечет вывод сведений о каждом коммите до одной строки (в ней будет хэш, и сообщение о коммите (здесь оно усекается утилитой Git)).
Ключ `--since`/ `--after` (это синонимы) выведет только коммиты, внесенные с указанной даты (дата задается в формате «`0000-00-00`», или прочем формате (подробнее о нем — в других источниках)).
Ключ `--untill/ --before` (это синонимы) выведет только коммиты, внесенные до указанной даты (дата задается в формате «`0000-00-00`», или прочем формате (подробнее о нем — в других источниках)).
Ключ `--autor` ограничит вывод команды коммитами, внесенными указанным автором (в значении ключа прописывается автор непосредственно/ без кавычек).
Ключ `--committer` ограничит вывод команды коммитами, внесенными указанным коммиттером (в значении ключа прописывается коммиттер непосредственно/ без кавычек).
Ключ `master` (можно указать целевую ветвь) выведет коммиты (достижимые из) целевой ветви.
Ключ `FileName` (можно указать целевой файл) выведет коммиты, в которых вносились изменения в указанный файл.
Ключ `--grep` выведет коммиты, содержащие в сообщении о коммите заданное слово (оно прописывается непосредственно, без кавычек; прочие способы задать это значение — см. другие источники).
Ключ `-S` выведет коммиты, в текстовых файлах которых содержится указанное слово (оно задается непосредственно, без кавычек, и приписывается к самому ключу; прочие способы задать это значение — см. другие источники).
Есть и прочие ключи — см. другие источники.

Отмена изменений

Отмена изменений выполняется командой `$ git commit -amend`. Эта команда фиксирует изменения, которые были staged, и добавляет их в последний (а не новый) коммит; если же изменилось только сообщение о commit, то изменится только это сообщение. Эта команда поддерживается, так как она может быть полезна при слишком рано выполненном commit (так как она позволяет не плодить лишние commit). И, разумеется, необходимо помнить, что команды `$ git commit -amend` и `$ git commit -a` никоим образом не являются синонимами, логика их действия разная.

Команда `$ git reset HEAD FileName` отменяет staging указанного файла; эта команда затрагивает только staging; однако, будучи вызванной с ключем `--hard`, она затронет и файл в рабочем каталоге.
Можно откатить изменения, внесенные после последней фиксации; это производится вызовом команды `$ git checkout -- FileName`; эта команда отменит изменения файла в рабочем каталоге.

Удаленные репозитории

Просмотр репозиториев

Удаленные репозитроии — это репозитории, находящиеся в сети. Они могут быть доступны на тех или ных правах.
Просмотр удаленных репозиториев (вывод их списка) осуществляется командой `$ git remote`. Эта команда выводит список коротких имен удаленных репозиториев пользователя; для того, чтобы вывести и его URL, используют ключ `-v` (тогда вывод команды приобретает вид: `RepName URL (fetch) …`).
Эта команда позволяет просмотреть подробности об удаленном репозитории, для этого она вызывается так:

```
$ git remote show RepName
```

в числе подробностей выводятся следующие сведения (см. справку).
Эта команда позволяет также удалить/переименовать удаленный репозиторий (переименовать - `$ git remote RepName NewRepName`.
Удалить ссылку на удаленный репозиторий можно командой `$ git remote rm RepName`.

Добавление удаленных репозиториев

Добавить (ссылку на) удаленный репозиторий можно командой:

```
$ git remote add RepName URL
```

в результате вызова этой команды будет добавлена ссылка на удаленный репозиторий; сам этот репозиторий будет доступен под именем `RepName`.

Скачивание с удаленных репозиториев

Для скачивания данных удаленного репозитория используют команду:

```
$ git fetch RepName
```

утилита `Git` экономит трафик — скачаны будут не все данные, а только те, что отсутствуют у пользователя. Скачивание производится в локальный репозиторий

(но слияния ветвей (скачанных и имеющихся локально) не производится). Скачанные ветви доступны локально как `RepName/BranchName`.
Есть аналогичная команда - `$ git pull`, она также выполняет слияние ветвей (подробности — далее в этом томе).
Эти команды могут использовать как короткие имена, так и адреса (но, разумеется, имена проще в записи).

Пополнение удаленных репозиториев

Команда `$ git push`, этой команде надо задать короткое имя (удаленного репозитория) и ветвь (локальную ветвь).

Переименование удаленных репозиториев

Переименование удаленных репозиториев (смена их коротких имен) производится командой:
```
$ git remote rename RepName NewRepName
```

Удаление удаленных репозиториев

Удаление (ссылки на) удаленный репозиторий производится командой:
```
$ git remote rm RepName
```

Теги

Теги — это метки версий. Они, как и сообщения о commit, и их хэши, представляют из себя сведения, идентифицирующие commit.

Вывод списка тегов

Команда `$ git tag`, о ключах команды — см. другие источники.

Создание тегов

Теги бывают легковесные (просто метка версии), и аннотированными (содержат различные пользовательские аннотации).
Создание аннотированных тегов выполняется следующей командой:
```
$ git tag -a VersionTag -m "TextOfComment"
```

для просмотра тега с аннотациями служит команда `$ git show VersionTag`.
Что касается легковесных тегов, то они аналогичны аннотированным (за тем исключением, что не позволяют работать с аннотациями).

Ветвления

Ветвления — это собственно ветвления линии разработки проекта. Таким образом, ветвь — это ни что иное, как секвенция коммитов. Коммиты ссылаются друг на друга, таким образом их секвенция имеет ссылочную целостность.
Основная ветвь разработки проекта по умолчанию носит имя `master`.
Создане новой ветви производится командой:

```
$ git branch NewBranchName
```

в результате этой команды будет создана новая ветвь, она будет брать начало с текущего коммита. `Git` знает, какой коммит — текущий (переменная `HEAD`).
После создания ветви работа по прежнему будет проистекать с текущей ветвью (а не новой). Чтобы перейти на целевую ветвь, ипользуется команда:

```
$ git checkout BranchName
```

целевая ветвь этой команды станет текущей ветвью (и это отобразится на переменной `HEAD`), и текущая ветвь окажется загружена в рабочий каталог. Есть сокращенная запись этой секвенции действий — создания ветви и перехода на нее (`$ git checkout -b BranchName`).
Операции над ветвями не требуют фиксации (однако, при работе с ветвями необходимо помнить, что ветвь — это секвенция коммитов, и если какие- либо данные должны быть включены в ветвь, то они сперва должны быть включены в commit).
Переход на целевую ветвь возможен только если текущая ветвь не изменялась, или же изменения фиксированы (собственно фиксированы, или отменены). Так или иначе, чтобы переход на целевую ветвь был возможен, состояние файлов в рабочей папке должно соответствовать состоянию их последней фиксации. Обеспечить это можно командами, рассмотренными выше в этом томе.

Слияние ветвей

Ветви могут быть слиты друг с другом (командой пользователя). Классический способ слияния — это такое слияние, при котором ветви (а это секвенции коммитов) сопоставляются в единую (в порядке следования); такой способ слияния принято называть “fast- forward”. Такое слияние выполняется командой:

```
$ git merge BranchName
```

по выполнению этой команды текущая ветвь будет слита с указанной.
Этот способ слияния применяется когда после ветвления одна из ветвей перестает пополняться коммитами. Только тогда этот способ применим. Если же после ветвления одна из ветвей (также) продолжает пополняться коммитами (как и другая), то применяется другой способ слияния - «слияние с исползованием merge commit». При этом способе слияния выполняется следующее: последние коммиты текущей и указанной ветвей используются как

предки для создания коммита слияния (он создается автоматически), этот коммит и будет последним коммитом текущей ветви в результате слияния ветвей. Этот способ слияния задействуется той же записью — конкретный способ слиянния выбирается самой `Git`.
При слиянии ветвей (каким- либо способом) возможны конфликты слияния. При конфликте слияния в слиянии будет отказано. Конфликт слияния возникает когда ветви имеют конфликтующие изменения (одни и те же файлы этих ветвей имеют разные изменения).

Удаление ветвей

В ходе работы с ветвями некоторые ветви могут устаревать, и оказываться в дальнейшем бесполезными. Удаление ветвей производится командой:
```
$ git branch -d BranchName
```

могут быть удалены только ветви, не являющиеся текущей ветвью. Кроме того, этой записью могут быть удалены только те ветви, данные из которых были добавлены (слиянием) в другие ветви (это предусмотрено во избежание утраты данных). Ветви, данные из которых не сливались, могут быть удалены альтернативной записью (`$ git branch -D BranchName`), различия в этих формах записей символические.

Список ветвей

Список ветвей может быть вызван командой (`$ git branch`). Текущая ветвь в списке ветвей обозначается псевдографикой (`*`), этот символ предшествует имени ветви. Ключ `--merged` этой команды означает включение в этот список только тех ветвей, данные из которых сливались в текущую ветвь, ключ `--no-merged` имеет противоположное действие. Если конкретнее, то этим командам можно задать целевую ветвь — на слияние с ней и будет выполняться проверка (тогда команда примет вид `--ComandName BranchName`), если целевая ветвь не задана — то по умолчанию это текущая ветвь.
Ключ `-v` этой команды приведет к выводу списка ветвей, где каждый пункт имеет вид: имя ветви, хэш последнего коммита ветви, коммент к нему.

Удаленные ветви

Удаленные ветви — это (ссылки на) ветви удаленных репозиториев. Они доступны (в записи) как `RepName/BranchName`.
Чтобы удаленные ветви были доступны, необходимо чтобы соответствующие удаленные репозитории были доступны (ссылки на них добавляются явно, командой `$ git remote add`).

При клонировании удаленного репозитория команда `$ git clone` (по умолчанию) присваивает имя `origin` клонируемому репозиторию. По этому (в различных обозначениях) это имя зачастую используется для обозначения удаленного репозитория (когда это — имя, а когда — просто обозначение, обычно бывает ясно из контекста).
Синхронизация с удаленным репозиторием осуществляется явно — командой `$ git fetch`.
Необходимо помнить, что эта команда работает не с самими данными, а лишь с ссылками на них (проще говоря, скачиваются не актуальные данные, а ссылки на актуальные данные). Даже по факту синхронизации удаленные ветви не доступны для редактирования; однако, они доступны для операции слияния — слияние производится характерной записью `$ git merge RepName/BranchName`. Чтобы удаленная ветвь стала доступна для редактирования, ее надо загрузить.
Загрузка удаленной ветви производится командой `$ git pull`. Эта команда в своей логике эквивалентна последовательности команд `$ git fetch`, и `$ git merge`.
Отправка локальной ветви на удаленный сервер производится явно — командой `$ git push`. Можно отправить ветвь и под другим именем — командой `$ git push RepName BranchName:NewBranchName`. При этом, если ветвь с данным именем уже есть, то данные будут отправлены в эту (уже существующую) ветвь.

Об отслеживании ветвей

Ветвь, созданная из удаленной ветви (тем или иным способом), является отслеживаемой. Последствия отслеживаемости следующие: команда `$ git push` может работать без аргументов (если текущей является отслеживаемая ветвь), команда `$ git pull` – тоже (и, кроме того, автоматически выполнит слияние данных ветвей).
При клонировании удаленных репозиториев ветвь `master` по умолчанию является отслеживаемой.
При необходимости, те или иные ветви (в том числе, ветвь `master`) могут быть настроены как отслеживаемые/ не отслеживаемые. Ветвь может быть настроена при создании — следующей командой:

```
$ git checkout -b BranchName RepName/BranchName
```

эта команда создаст ветвь из удаленной ветви. У этой команды есть синонимичная запись: `$ git checkout --track RepName/BranchName`.
Ветвь может быть настроена также после создания — следующей командой:

```
$ git branch -u RepName/BranchName
```

эта команда сделает указанную удаленную ветвь отслеживаемой одноименной ей локальной ветвью.
Просмотреть список отлеживаемых ветвей можно следующей командой:

```
$git branch -vv
```

эта команда выведет список в следующем виде:
```
BranchName Hash [RepName/BranchName: ahead M, behind N] CommitMessage
```

в этой записи `ahead M` означает «локальная ветвь содержит `M` не отправленных на `RepName` коммитов», `behind N` означает «`RepName` имеет `N` не скачанных коммитов». Прочие обозначения не требуют комментариев. Эта команда выводит также и не отслеживаемые ветви, они в этом списке выводятся в виде `BranchName Hash CommitMessage`. Эта команда не работает с сервером (а вместо этого пользуется закэшированными при обращении к серверу данными), поэтому она может сообщать не актуальные данные — чтобы данные были актуальны, стоит воспользоваться командой `$ git fetch –all`.

Удаление удаленных ветвей

Удаленная ветвь может быть удалена (из удаленного репозитория). Это производится следующей командой:
```
$ git push RepName --delete BranchName
```

в результате указанная ветвь будет удалена с удаленного репозитория.

Перебазирование ветвей

Перебазирование ветвей — это операция, аналогичная слиянию (но, в отличии от слияния, она приводит не к копированию данных, а к их перемещению).
Перебазирование производится следующей командой:
```
$ git rebase BranchName
```

в результате текущая ветвь перебазируется в указанную (именно так!).
Команда перебазирования зарекомендовала себя проблематичной в применении.
Перебазирование должно выполняться только в локальном репозитории.
Прочие сведения — в прочих источниках.

Часть 4 (Additional ref')

Commit message

Сообщение о коммите следует оформлять следующим образом:
строка заголовка сообщения — это тезис, описывающий суть коммита (следует ограничиться 50 символами);
пустая строка (не следует ей пренебрегать);
тело сообщения о коммите — это комментарий к тезису в заголовке сообщения о коммите (следует ограничиться 72 символами);

этот формат следует соблюдать, иначе различные команды могут счесть сообщение о коммите не приемлемым (соответственно, и сам коммит).

О пользовании Git

В процессе использования Git может обнаружиться следующее обстоятельство — ряд пользователей одновременно пытается отправить свои коммиты на удаленный репозиторий. При этом отправка данных осуществляется в порядке очереди. Данные удастся отправить лишь первому в этой очереди. Остальные пользователи, желающие отправить данные, претерпят фиаско. Им потребуется скачать уже обновленные данные с удаленного репозитория (при этом, можно скачивать как все, так и только обновленные ветви), провести слияние с ними, и только потом отправить свои данные. Конкретный проект, при этом, может потребовать выполнить тестирование после слияния (и в этом есть смысл). Таков процесс использования Git.
Следует дополнить, что у пользователя могут быть права на внесение изменений в удаленный репозиторий проекта, а могут и не быть — тогда наработанные изменения следует отправлять по электронной почте пользователю, наделенному такими правами (кстати говоря, это устоявшаяся практика в работе над крупными публичными проектами). Разумеется, при этой практике (обычно) высылается лишь сообщение о завершении наработок (так как уполномоченный пользователь может просто выполнить слияние с этими наработками).

О команде push

Команде `$ git push` доступен также следующий синтаксис:
`$ git push -u RepName LocBranshName:RemBranchName`

флаг `-u` отвечает за свою функциональность; u это от upstream, соответственно, флаги `-u` и `upstream` это синонимы.

Важнейшая особенность Git

Утилита Git, как и говорилось в томе Git (данном томе), предназначена для работы с исходными кодами — бинарные файлы не открываются этой утилитой, и не подвергаются слиянию. При работе с текстовыми файлами эта утилита проверяет их на наличие конфликтов слияния; но, конечно же, не проверяет на предмет синтаксической правильности (с точки зрения синтаксиса используемых языков) — поэтому, даже при корректном слиянии, работоспособность кодов проекта может быть нарушена. Поэтому, работа над кодом — это задача именно программирования (а не эксплуатации Git), и редактирование кода должно осуществляться именно в IDE. Никогда не следует забывать об этой особенности.

О поддержке проекта

Поддержка проекта, во многом, заключается в деятельности пользователя, уполномоченного вносить наработки в удаленный репозиторий проекта.
Даже если принято решение о внесении этих наработок, то (обычно) они предварительно вносятся в тематическую ветвь, а внесение их в основную ветвь проекта производится позже — когда это предписывается проектом.

О сайте GitHub

Сайт GitHub предлагает услугу хостинга репозиториев (и в настоящее время является самым популярным сайтом в своем роде). Им пользуются как индивидуальные разработчики, так и корпоративные разработчики.

О пользовании сайтом

Пользование сайтом (в деталях) изменяется в соответствии с изменениями дизайна сайта.
Общая же схема следующая.
Пользование этим сайтом начинается с заведения аккаунта на нем.
Далее следует выбрать тариф, по которому будет использоваться услуга.
Для того, чтобы получить локальную копию удаленного проекта, необходимо нажать кнопку fork (в верху экрана).
По наработке изменений следует отправить запрос на их внесение (если нет полномочий вносить их самому). Отправка запроса производится так —

отправляется ветвь (находящаяся в ведении пользователя), и становится доступна кнока Pull request. При щелчке по ней откроется окно — в него вводится комментарий запроса. Создание запроса завершается щелчком по кнопке Create pull request. По щелчку по ней происходит отправка запроса. Следует помнить, что доработка изменений возможна и после отправки запроса (но, конечно, это следует практиковать только при необходимости такой доработки, и только в пределах одной команды — чтобы доработки были согласованными).
Владелец проекта принимает решение о слиянии наработок, или же просто закрытии запроса. Каждая такая задача выполняется соответствующей кнопкой.
Прочие детали эксплуатации GitHub – это специфика дизайна этого сайта.

О прочих вопросах

О использовании хэшей

Как и говорилось в томе Git (данном томе), ветви можно указывать как по именам, так и по хэшу. Хэши вычислияются самой утилитой Git (по алгоритму SHA1). При вычилении хэшей по этому алгоритму в ряде случаев происходят коллизии — разные входные данные алгоритма результируют одним и тем же значением хэш- суммы.
Возникновение коллизий маловероятно, однако возможно. В случае коллизии она не разрешается (!). И тогда у соответствующих файлов будут одинаковые хэши. Тогда обращение к ветви по хэшу приведет к обращению к первому найденному результату.

О диапазонах коммитов

Коммиты могут быть заданы также диапазонами.
Запись `BranchName0..BranchName1` эквивалентна следующей записи (ее нотацию следует трактовать обычным для таких записей способом — как в математике, при обозначении промежутков) — (`BranchName0`, `BranchName1`] – по этой записи доступны коммиты, которые из ветви `BranchName1` и идут до ветви `BranchName0` не включительно. В этой записи одну из ветвей можно опустить — тогда используется значение `HEAD`.
Запись `BranchName0...BranchName1` эквивалентна следующей записи (ее нотацию следует трактовать обычным для таких записей способом — как в формальной логике, при обозначении логических операций) — `BranchName0 XOR BranchName1` – по этой записи доступны коммиты, входящие в указанные ветви, за исключением точки ветвления.
Эти записи доступны в командах, которым можно задать ряд ветвей.

Скрытие данных

Скрытие данных осуществляется командой `$ git stash`. Будучи вызванной без аргументов, эта команда заталкивает в стек (предназначенный для хранения этих данных) все (незафиксированные) изменения (отслеживаемых) файлов рабочего каталога. Каждый вызов этой команды пополняет стек, если его есть чем пополнить. Команда `$ git stash --keep-index` не затрагивает файлы, которые staged.
Просмотреть стек можно командой:
`$ git stash list`

эта команда выведет содеримое этого стека, где каждый «кадр» имеет вид:
`stash@{№}: WIP on BranchName Hash CommitMessage`

каждый «кадр» доступен по записи `stash@{№}`.
«Кадры» выталкиваются в рабочий каталог командой `$ git stash apply`, этой команде можно указать целевой кадр (записью `stash@{№}`), верхушку стека можно указать проще — опустив явное указание. Строго говоря, эта команда не выталкивает данные, а копирует их из стека в рабочий каталог. Эта команда не затрагивает состояния файлов (если их потребутся сделать staged, то это делается явно — командой `$ git add`, или же `$ git stassh apply --index`). Команда `$ git stash pop` приводит к выталкиванию (именно выталкиванию) из стека в рабочий каталог. Эти команды (неявно) производят слияние с контентом рабочего каталога (и, в интересах их применения, конфликтов быть не должно).
Команда `$ git stash branch BranchName` создает новую ветвь с указанным именем (ветвь создается на базе последнего коммита ветви, на которой произошел вызов команды), выталкивает в нее вершину стека (поведение этой команды следует уточнить для актуальной версии утилиты).
Удалить целевой «кадр» из стека можно командой `$ git stash drop`.

Очистка рабочего каталога

Очистка рабочего каталога выполняется командой `$ git clean`.
Поведение этой команды следует уточнить для актуальной версии утилиты.


Git