Глава 5. Повседневное использование Mercurial

Содержание

5.1. Указание Mercurial, какие файлы необходимо отслеживать

Mercurial не работает с файлами в хранилище, пока вы не скажете ему, чтобы он управлял ими. Команда *hg status* покажет о каких файлах Mercurial не знает, он использует «`?`» для отображения таких файлов.

Чтобы сказать Mercurial отслеживать файлы, используйте команду *hg add*. После того, как вы добавили файл, запись в результатах *hg status* для этого файла изменится с «`?`» на «`A`».

```
$ hg init add-example
```

```
$ cd add-example
```

```
$ echo a > myfile.txt
```

```
$ hg status
```

```
? myfile.txt
```

```
$ hg add myfile.txt
```

```
$ hg status
```

```
A myfile.txt
```

```
$ hg commit -m 'Added one file'
```

```
$ hg status
```

После запуска *hg commit* файлы, которые вы добавили перед фиксацией не будут отображаться в выводе *hg status*. Дело в том, что *hg status* по умолчанию сообщает только о «интересных» файлах — о тех, что вы модифицировали, либо указали Mercurial'у сделать что-либо с ними. Если ваш репозиторий содержит тысячи файлов, то вам редко понадобится информация обо всех не измененных файлах, которые отслеживает Mercurial. (Вы можете узнать и о них; мы вернемся к этому позже.)

Когда вы добавляете файл, Mercurial сразу ничего с ним не делает. Только после фиксации Mercurial сделает снимок состояния файла. Он будет продолжать отслеживать изменения каждый раз после фиксации, до тех пор пока файл не будет удален.

5.1.1. Явное и неявное именование файлов

При выполнении любой команды, если вы не указываете имя файла, Mercurial понимает это как «Я собираюсь работать со всеми файлами в этом каталоге и его подкаталогах».

```
$ mkdir b
```

```
$ echo b > b/somefile.txt
```

```
$ echo c > b/source.cpp
```

```
$ mkdir b/d
```

```
$ echo d > b/d/test.h
```

```
$ hg add b
```

```
adding b/d/test.h
```

```
adding b/somefile.txt
```

```
adding b/source.cpp
```

```
$ hg commit -m 'Added all files in
subdirectory'
```

Обратите внимание, что в данном примере в отличие от предыдущего Mercurial вывел имена добавленных файлов.

Когда, как в предыдущем случае, мы явно указываем имя файла добавляемого в командной строке. Mercurial делает предположение, что вы знаете, что делаете, и не выводит ничего.

Однако, когда мы *подразумеваем* файлы, указывая только имя каталога, Mercurial дополнительно выводит имя каждого обработанного файла. Это делает процесс более прозрачным и уменьшает вероятность незаметных неприятных сюрпризов. Такое поведение свойственно большинству команд Mercurial.

5.1.2. Mercurial отслеживает файлы, а не каталоги

Mercurial не отслеживает информацию о каталогах. Вместо этого он отслеживает путь к файлу. Перед созданием файла он создает недостающие каталоги пути. После удаления — удаляет все пустые каталоги, которые присутствовали в пути файла. Кажется в этом нет ничего особенного, но имеется незначительное следствие: в Mercurial невозможно содержать пустой каталог.

Пустые каталоги нужны не часто, и есть несколько вариантов обойти ограничение, чтобы достичь желаемого эффекта. Разработчики Mercurial полагали, что усложнения, которые потребуются для поддержки управления пустыми каталогами, не стоят незначительных преимуществ данной опции.

Если вам требуется пустой каталог в репозитории, есть несколько путей сделать это. Первый — создать каталог и добавить в него *hg add* скрытый («hidden») файл. В Unix-подобных системах любой файл, имя которого начинается с точки («.»), рассматривается как скрытый большинством команд и инструментами GUI. Этот метод представлен ниже.

```
$ hg init hidden-example
```

```
$ cd hidden-example
```

```
$ mkdir empty
```

```
$ touch empty/.hidden
```

```
$ hg add empty/.hidden
```

```
$ hg commit -m 'Manage an empty-looking
directory'
```

```
$ ls empty
```

```
$ cd ..
```

```
$ hg clone hidden-example tmp
```

```
updating to branch default
```

```
1 files updated, 0 files merged, 0 files
removed, 0 files unresolved
```

```
$ ls tmp
```

```
empty
```

```
$ ls tmp/empty
```

Другой вариант удовлетворить потребность в пустой директории — просто создать необходимый каталог автоматически с помощью скриптов.

5.2. Как прекратить отслеживание файла

Когда файл больше не нужен в репозитории, используйте команду *hg remove*, которая удаляет файл и указывает Mercurial прекратить его отслеживание. Удаленный файл в выводе *hg status* отображается буквой «R».

```
$ hg init remove-example
```

```
$ cd remove-example
```

```
$ echo a > a
```

```
$ mkdir b
```

```
$ echo b > b/b
```

```
$ hg add a b
```

```
adding b/b
```

```
$ hg commit -m 'Small example for file removal'
```

```
$ hg remove a
```

```
$ hg status
```

```
R a
```

```
$ hg remove b
```

```
removing b/b
```

После выполнения *hg remove* над файлом Mercurial больше не отслеживает его изменения даже если вы пересоздадите файл с таким же именем в этом каталоге. Если вы создали одноименный файл и хотите, чтобы Mercurial отслеживал новый файл, просто выполните *hg add* с ним. Mercurial будет знать что вновь добавленный файл никак не связан со старым одноименным файлом.

5.2.1. Удаление файла не влияет на его историю

Важно понимать, что удаление файла, имеет только два результата.

- Удаляется текущая версия файла из рабочего каталога.
- Mercurial прекращает отслеживать изменения над файлом со следующей фиксации (commit'a)

Удаление файла в любом случае *не* меняет *историю* его изменений.

Если вы обновите рабочий каталог до версии, в которой удаленный файл еще отслеживался, то он появится в каталоге и будет содержать данные, которые в нем были на момент фиксации версии. Если вы обновите каталог до более поздней версии, где данный файл уже был удален, Mercurial снова удалит файл из каталога.

5.2.2. Отсутствующие файлы

Mercurial считает *потерянными* файлы, которые вы удалили не используя *hg remove*. Отсутствующие файлы в выводе *hg status* отображаются с «!». Команды Mercurial как правило ничего не сделают с потерянными файлами.

```
$ hg init missing-example
```

```
$ cd missing-example
```

```
$ echo a > a
```

```
$ hg add a
```

```
$ hg commit -m 'File about to be missing'
```

```
$ rm a
```

```
$ hg status
```

```
! a
```

Если в вашем репозитории есть файл, который *hg status* отображает как потерянный, и вы хотите его действительно удалить, вы можете это сделать командой *hg remove* *--after*.

```
$ hg remove --after a
```

```
$ hg status
```

```
R a
```

С другой стороны если вы случайно удалили файл, используйте команду *hg revert* filename чтобы его восстановить. Файл будет восстановлен в неизмененной форме.

```
$ hg revert a
```

```
$ cat a
```

```
a
```

```
$ hg status
```

5.2.3. Замечание: почему в Mercurial явно указывается удаление файла?

Возможно вы удивитесь, что Mercurial требует явно указывать удаление файла. Раньше при разработке Mercurial можно было удалять файл, когда угодно. Mercurial замечал отсутствие файла автоматически при следующем запуске *hg commit* и прекращал отслеживание файла. На практике выяснилось, что это приводит к случайному незаметному удалению файлов.

5.2.4. Полезное сокращение — добавление и удаление файлов в один прием

Mercurial предоставляет комбинированную команду *hg addremove*, которая добавляет неотслеживаемые файлы и помечает отсутствующие файлы как удаленные.

```
$ hg init addremove-example
```

```
$ cd addremove-example
```

```
$ echo a > a
```

```
$ echo b > b
```

```
$ hg addremove
```

```
adding a
```

```
adding b
```

Команда *hg commit* также имеет опцию `-A`, которая выполняет то же самое добавление-и-удаление, за которыми сразу же следует фиксация.

```
$ echo c > c
```

```
$ hg commit -A -m 'Commit with addremove'
```

```
adding c
```

5.3. Копирование файлов

Для создания копии файла Mercurial предоставляет команду *hg copy*. Когда вы копируете файл с помощью этой команды, Mercurial создает запись о том, что новый файл является копией исходного файла. Он использует такие скопированные файлы, когда вы объединяете свою работу с чьей-либо еще.

5.3.1. Поведение копии при слиянии

При объединении получается, что изменения «преследуют» копии. Чтобы лучше продемонстрировать эту мысль, рассмотрим пример. Для начала возьмем обычный скромный репозиторий с одним файлом.

```
$ hg init my-copy
```

```
$ cd my-copy
```

```
$ echo line > file
```

```
$ hg add file
```

```
$ hg commit -m 'Added a file'
```

Нам необходимо работать параллельно, и затем объединить. Поэтому давайте клонируем наш репозиторий.

```
$ cd ..
```

```
$ hg clone my-copy your-copy
```

```
updating to branch default
```

```
1 files updated, 0 files merged, 0 files
removed, 0 files unresolved
```

Возвращаясь к исходному репозиторию, выполним команду *hg copy*, чтобы сделать копию первого созданного файла.

```
$ cd my-copy
```

```
$ hg copy file new-file
```

Если посмотрим вывод команды *hg status*, увидим, что скопированный файл отображается как обычный добавленный файл.

```
$ hg status
```

```
A new-file
```

Но если мы укажем опцию `-C` в команде *hg status*, в выводе будет еще одна строка: файл, копия которого была сделана.

```
$ hg status -C
```

```
A new-file
```

```
  file
```

```
$ hg commit -m 'Copied file'
```

Теперь, вернувшись к клонированному репозиторию, сделаем параллельные изменения. Добавим строку в исходный файл.

```
$ cd ../your-copy
```

```
$ echo 'new contents' >> file
```

```
$ hg commit -m 'Changed file'
```

Теперь мы имеем измененный файл *file* в этом репозитории. Когда мы подтягиваем изменения из первого репозитория и объединяем две последние ревизии (head), Mercurial переносит изменения, которые были сделаны в файле *file* в его копию *new-file*.

```
$ hg pull ../my-copy
```

```
pulling from ../my-copy
```

```
searching for changes
```

```
adding changesets
```

```
adding manifests
```

```
adding file changes
```

```
added 1 changesets with 1 changes to 1 files
(+1 heads)
```

```
(run 'hg heads' to see heads, 'hg merge' to
merge)
```

```
$ hg merge
```

```
merging file and new-file to new-file
```

```
0 files updated, 1 files merged, 0 files
removed, 0 files unresolved
```

```
(branch merge, don't forget to commit)
```

```
$ cat new-file
```

```
line
```

```
new contents
```

5.3.2. Почему изменения следуют за копией?

Перенос изменений исходного файла в копии может показаться понятным лишь посвященным, но в большинстве случаев это крайне желательно.

Прежде всего напомним, что подобный перенос происходит *только* при объединении. И если вы делаете *hg copy* файла и последовательно изменяете оригинал в процессе работы, с копией ничего не происходит.

Во-вторых, надо понимать, что изменения будут вноситься в копию до тех пор пока репозиторий, из которого подтягиваются изменения, *не знает* о существовании копии.

Mercurial делает это по следующей причине. Представим, что я исправляю серьезный баг в исходнике, и фиксирую изменения. А в то же время вы решаете сделать *hg copy* файла в вашем репозитории, при этом вы ничего не знаете о баге и не видите последней фиксации, и вы начинаете колдовать со своей копией файла.

Когда вы подтянули и объединили мои изменения, и Mercurial *не* перенес изменения в копии, ваш исходник будет содержать баг, и пока вы не вспомните и не перенесете исправление вручную, баг *останется* не исправленным в вашей копии файла.

Благодаря автоматическому переносу зафиксированных изменений от исходного файла к копиям, Mercurial предотвращает подобные проблемы. Насколько мне известно, Mercurial *единственная* система контроля версий, которая подобным образом переносит изменения в копии.

Когда в истории изменений есть запись о появлении копии и последующего объединения, обычно не требуется последующий перенос изменений в исходном файле в копию, и поэтому Mercurial переносит только те изменения в копию, которые были до этой точки, а не дальнейшие.

5.3.3. Как сделать, чтобы изменения *не* преследовали копию

Если по каким-то причинам вы решили, что дело автоматического переноса изменений в копии не для Вас, то просто используйте обычную системную команду копирования (в Unix-подобных системах, это *cp*), а затем добавьте файл вручную с помощью *hg add*. Перед тем как это сделать, перечитайте Раздел 5.3.2, «Почему изменения следуют за копией?», и убедитесь, что в вашем специфическом случае это действительно не нужно.

5.3.4. Поведение команды *hg copy*

При выполнении команды *hg copy* Mercurial делает копии каждого исходного файла в таком виде, в каком на текущий момент он находится в рабочем каталоге. Это означает, что если вы вносите изменения в файл, а

затем делаете его копию без предварительной фиксации изменений, то копия будет содержать изменения, внесенные до момента копирования. (Мне кажется такое поведение нелогичным, поэтому я обращаю внимание здесь.)

Действие команды *hg copy* подобно команде *cp* в Unix (для удобства вы можете использовать алиас *hg cp*). Последний аргумент — адрес *назначения*, все предыдущие — *источники*.

Если вы указываете единственный файл как источник и файл назначения не существует, будет создан новый файл с этим именем.

```
$ mkdir k
```

```
$ hg copy a k
```

```
$ ls k
```

```
a
```

Если адрес назначения указан каталог, Mercurial сделает копии источников в этот каталог

```
$ mkdir d
```

```
$ hg copy a b d
```

```
$ ls d
```

```
a  b
```

Копирование каталогов рекурсивно, и сохраняет структуру каталогов источника.

```
$ hg copy z e
```

```
copying z/a/c to e/a/c
```

Если и источник, и назначение — каталоги, то дерево каталогов источника будет также создано в каталоге назначения.

```
$ hg copy z d
```

```
copying z/a/c to d/z/a/c
```

Также как с командой *hg remove*, если вы создали копии вручную и хотите, чтобы Mercurial знал, что это копии, просто используйте опцию `--after` в команде *hg copy*.

```
$ cp a n
```

```
$ hg copy --after a n
```

5.4. Переименование файлов

Переименование файлов обычно используется чаще, чем копирование. Я рассмотрел команду *hg copy* до команды переименования, так как по существу Mercurial воспринимает переименование так же, как копирование. Следовательно, понимание того, как ведет себя Mercurial с копиями, объяснит, чего ожидать от переименования файлов.

При выполнении команды *hg rename* Mercurial копирует исходные файлы, затем удаляет их и помечает как удаленные.

```
$ hg rename a b
```

Команда *hg status* показывает, что новые файлы были добавлены, а файлы, с которых они были скопированы, удалены.

```
$ hg status
```

```
A b
```

```
R a
```

Так же как с результатами *hg copy* мы должны использовать опцию `-C` команды *hg status*, чтобы увидеть, что добавленный файл действительно отслеживается Mercurial, как исходный, уже удаленный, файл.

```
$ hg status -C
```

```
A b
```

```
  a
```

```
R a
```

Так же как и для команд *hg remove* и *hg copy*, вы можете указать Mercurial о переименовании после выполнения, используя опцию `--after`. В большинстве других случаев, поведение команды *hg rename* и ее поддерживаемых опций подобно поведению команды *hg copy*.

Если вы знакомы с командной строкой Unix, вы будете рады узнать, что команда *hg rename* может работать как *hg mv*.

5.4.1. Переименование файлов и объединение изменений

С тех пор как переименование в Mercurial реализовано как копирование и удаление, после переименования происходит такой же перенос изменений при объединении (merge), как и после копирования.

Если вы изменили файл и переименовали его, а затем мы объединяем наши изменения, мои модификации в файле с первоначальным именем будут перенесены в файл с новым именем. (Это кажется достаточно простым действием, однако не во всех системах контроля версий оно работает.)

Если по поводу следования изменений за копией вы возможно кивнете и скажете: «Да, это может быть полезно», то, что касается следования их при переименовании, абсолютно ясно, что это действительно важно. Без этой возможности изменения легко бы оставались осиротевшими при переименовании файлов.

5.4.2. Расходящиеся переименования и слияние

Расходящиеся переименования происходят, когда два разработчика берутся за один файл — назовем его *foo* в своих репозиториях.

```
$ hg clone orig anne
```

```
updating to branch default
```

```
1 files updated, 0 files merged, 0 files
removed, 0 files unresolved
```

```
$ hg clone orig bob
```

```
updating to branch default
```

```
1 files updated, 0 files merged, 0 files
removed, 0 files unresolved
```

Анна переименовывает файл в *bar*.

```
$ cd anne
```

```
$ hg rename foo bar
```

```
$ hg ci -m 'Rename foo to bar'
```

Тем временем Bob переименовывает его в *quux*. (Помните, что *hg mv* является псевдонимом для *hg rename*.)

```
$ cd ../bob
```

```
$ hg mv foo quux
```

```
$ hg ci -m 'Rename foo to quux'
```

Я расцениваю это как конфликт, потому что каждый разработчик выразил свое мнение о том, как данный файл должен называться.

Как вы считаете, что должно произойти когда они объединят работу? На самом деле Mercurial всегда сохраняет *оба* имени при слиянии изменений, которые содержат расходящиеся переименования.

```
# See
http://www.selenic.com/mercurial/bts/issue455
```

```
$ cd ../orig
```

```
$ hg pull -u ../anne
```

```
pulling from ../anne
```

```
searching for changes
```

```
adding changesets
```

```
adding manifests
```

```
adding file changes
```

```
added 1 changesets with 1 changes to 1 files
```

```
1 files updated, 0 files merged, 1 files
removed, 0 files unresolved
```

```
$ hg pull ../bob
```

```
pulling from ../bob
```

```
searching for changes
```

```
adding changesets
```

```
adding manifests
```

```
adding file changes
```

```
added 1 changesets with 1 changes to 1 files
(+1 heads)
```

```
(run 'hg heads' to see heads, 'hg merge' to
merge)
```

```
$ hg merge
```

```
note: possible conflict - foo was renamed
multiple times to:
```

```
 bar
```

```
 quux
```

```
1 files updated, 0 files merged, 0 files
removed, 0 files unresolved
```

```
(branch merge, don't forget to commit)
```

```
$ ls
```

```
bar  quux
```

Обратите внимание, что Mercurial предупредил о наличии расходящихся переименований, но оставил на ваше усмотрение, как с ними поступить после слияния.

5.4.3. Сходящиеся переименования и слияние

Другой вариант конфликта возникает, когда два человека переименовывают два разных файла и дают им одно и то же имя. В таком случае Mercurial выполняет стандартное слияние и предоставляет вам управлять им для нахождения подходящего решения.

5.4.4. Другие проблемы с именованием

У Mercurial есть давнишний баг, который дает ошибку при выполнении слияния, если с одной стороны имеется некоторый файл, а с другой стороны — каталог с таким же именем. Баг задокументирован как Mercurial issue 29.

```
$ hg init issue29
```

```
$ cd issue29
```

```
$ echo a > a
```

```
$ hg ci -Ama
```

```
adding a
```

```
$ echo b > b
```

```
$ hg ci -Amb
```

```
adding b
```

```
$ hg up 0
```

```
0 files updated, 0 files merged, 1 files
removed, 0 files unresolved
```

```
$ mkdir b
```

```
$ echo b > b/b
```

```
$ hg ci -Amc
```

```
adding b/b
```

```
created new head
```

```
$ hg merge
```

```
abort: Is a directory:
/tmp/issue29iLXJ9A/issue29/b
```

5.5. Избавление от ошибок

У Mercurial есть несколько полезных команд для восстановления после некоторых распространенных ошибок.

Команда *hg revert* позволяет отменить изменения, сделанные в рабочем каталоге. Например, если вы случайно выполнили *hg add*, просто запустите *hg revert*, указав имя добавленного файла, и файл больше не будет считаться добавленным для отслеживания в Mercurial. *hg revert* можно использовать и для отмены от ошибочных изменений в файле.

Необходимо помнить, что команда *hg revert* действует только на изменения, которые не были фиксированы. Если вы зафиксировали изменение, но поняли, что произошла ошибка, вы по-прежнему можете ее исправить, хотя возможности будут более ограничены.

Дополнительная информация о команде *hg revert* и о том, что можно сделать с зафиксированными изменениями, приведена в Глава 9, *Поиск и исправление ваших ошибок*.

5.6. Работа со сложными слияниями

В сложных и крупных проектах, не редкость слияние двух ревизий является головной болью. Предположим, что существует большой исходный файл, который был сильно отредактирован каждой стороной слияния: это практически неминуемо приведет к конфликтам, некоторые из которых могут потребовать несколько попыток разобраться.

Давайте начнём с простого случая, чтоб увидеть как с этим бороться. Начнем с репозитория, содержащего один файл, и клонируем его дважды.

```
$ hg init conflict
```

```
$ cd conflict
```

```
$ echo first > myfile.txt
```

```
$ hg ci -A -m first
```

```
adding myfile.txt
```

```
$ cd ..
```

```
$ hg clone conflict left
```

```
updating to branch default
```

```
1 files updated, 0 files merged, 0 files
removed, 0 files unresolved
```

```
$ hg clone conflict right
```

```
updating to branch default
```

```
1 files updated, 0 files merged, 0 files
removed, 0 files unresolved
```

В одном клоне, будем изменять файл по одному пути.

```
$ cd left
```

```
$ echo left >> myfile.txt
```

```
$ hg ci -m left
```

В другом клоне, будем изменять файл по одному пути.

```
$ cd ../right
```

```
$ echo right >> myfile.txt
```

```
$ hg ci -m right
```

Далее вытянем каждую ревизию в наш первоначальный репозиторий.

```
$ cd ../conflict
```

```
$ hg pull -u ../left
```

```
pulling from ../left
```

```
searching for changes
```

```
adding changesets
```

```
adding manifests
```

```
adding file changes
```

```
added 1 changesets with 1 changes to 1 files
```

```
1 files updated, 0 files merged, 0 files
removed, 0 files unresolved
```

```
$ hg pull -u ../right
```

```
pulling from ../right
```

```
searching for changes
```

```
adding changesets
```

```
adding manifests
```

```
adding file changes
```

```
added 1 changesets with 1 changes to 1 files
(+1 heads)
```

```
not updating: crosses branches (merge branches
or update --check to force update)
```

Мы думаем, что теперь в нашем репозитории находится две головных ревизии.

```
$ hg heads
```

```
changeset:   2:ee01e5caae49
```

```
tag:         tip
```

```
parent:      0:07ad45cd9c8f
```

```
user:        Bryan O'Sullivan
<bos@serpentine.com>
```

```
date:        Thu Feb 02 14:09:37 2012 +0000
```

```
summary:     right
```

```

```

```
changeset:   1:0a7bacc4a7aa
```

```
user:        Bryan O'Sullivan
<bos@serpentine.com>
```

```
date:        Thu Feb 02 14:09:37 2012 +0000
```

```
summary:     left
```

```

```

Обычно, если мы запустим *hg merge* в этот момент, он перенаправит нас в графический интерфейс, который позволит нам решать вручную противоречия в изменениях в файле *myfile.txt*. Однако, чтобы упростить эту задачу для представления здесь, мы хотели бы чтоб слияние провалилось. Вот один из способов, которым мы можем этого добиться.

```
$ export HGMERGE=false
```

Мы сообщили механизму слияния Mercurial выполнить команду *false* (которая, как мы и хотели, потерпит неудачу) если обнаружит, что слияние не может разрешится автоматически.

Если мы сейчас запустим *hg merge*, он должен остановится, и сообщить о провале.

```
$ hg merge
```

```
merging myfile.txt
```

```
merging myfile.txt failed!
```

```
0 files updated, 0 files merged, 0 files
removed, 1 files unresolved
```

```
use 'hg resolve' to retry unresolved file
merges or 'hg update -C .' to abandon
```

Даже если мы не заметим, что объединение не удалось, Mercurial не даст нам случайно зафиксировать результат неудачного слияния.

```
$ hg commit -m 'Attempt to commit a failed
merge'
```

```
abort: unresolved merge conflicts (see hg help
resolve)
```

Когда *hg commit* выполняется неудачно, как в этом случае, можно предположить, что мы используем неправильную команду *hg resolve*. Как обычно, *hg help resolve* выведет полезную справку.

5.6.1. Файл анализа состояний

Когда происходит слияние, большая часть файлов, как правило, остаётся без изменений. Для каждого файла, в котором Mercurial что-то делает, он отслеживает состояние файла.

- *Разрешенные* файлы, которые был успешно объединены, либо автоматически с помощью Mercurial или вручную с помощью человеческого вмешательства.
- *Неразрешенные* файлы — не были объединены успешно, необходимо уделить дополнительное внимание.

Если Mercurial видит *любой* файл в неразрешенных состояние после слияния, он будет считать, что слияние не увенчались успехом. К счастью, нам не запускать всё слияние с нуля.

Опция `--list` или `-l` команды *hg resolve* печатает состояние каждого объединяемого файла.

```
$ hg resolve -l
```

```
U myfile.txt
```

В выводе *hg resolve*, разрешённые файлы отмечены `R`, а неразрешённые файлы отмечены `U`. Если какие-либо файлы перечислены с `U`, мы знаем, что попытка зафиксировать результаты слияния не удастся.

5.6.2. Разрешение файлов при слиянии

Есть несколько вариантов, чтобы переместить файл из неразрешенного в разрешенное состояние. Самым распространенным является перезапуск *hg resolve*. Если мы будем пропускать отдельные имена файлов или каталогов, он будет повторять слияние неразрешенных файлов в этих местах. Мы также можем использовать опцию `--all` или `-a`, которая заставит повторить слияние для *всех* нерешенных файлов.

Mercurial также позволяет нам изменять состояние файла напрямую. Мы можем вручную пометить файл как решённый с помощью опции `--mark`, или, как нерешенный используя опцию `--unmark`. Это позволяет очистить особенно грязные сторонние дополнения, и следить за каждым файлом в прогрессе выполнения.

5.7. Более удобные diff-ы

Вывода команды *hg diff* по-умолчанию обратно совместимым с обычной командой *diff*, но имеет некоторые недостатки.

Рассмотрим случай, когда мы используем *hg rename* для переименования файла.

```
$ hg rename a b
```

```
$ hg diff
```

```
diff -r e1f8689c7be4 a
```

```
--- a/a Thu Feb 02 14:09:36 2012 +0000
```

```
+++ /dev/null   Thu Jan 01 00:00:00 1970 +0000
```

```
@@ -1,1 +0,0 @@
```

```
-a
```

```
diff -r e1f8689c7be4 b
```

```
--- /dev/null   Thu Jan 01 00:00:00 1970 +0000
```

```
+++ b/b Thu Feb 02 14:09:36 2012 +0000
```

```
@@ -0,0 +1,1 @@
```

```
+a
```

Вывод *hg diff* выше скрывает тот факт, что мы просто переименовали файл. Команда *hg diff* принимает опции `--git` или `-g`, чтобы использовать новый формат diff, который отображает такую информацию в более читаемом виде.

```
$ hg diff -g
```

```
diff --git a/a b/b
```

```
rename from a
```

```
rename to b
```

Эта функция также помогает в случаях, которые в противном случае могут ввести в заблуждение: файл, который, как представляется, измененился в соответствии с *hg status*, но для которого *hg diff* ничего не выводит. Такая ситуация может возникнуть, если мы изменили исполняемый файл.

```
$ chmod +x a
```

```
$ hg st
```

```
M a
```

```
$ hg diff
```

Обычная команда *diff* не обращает внимания на права доступа к файлу, поэтому *hg diff* ничего по умолчанию не выводит. Если мы запускаем ее с опцией `-g`, она расскажет нам, что происходит.

```
$ hg diff -g
```

```
diff --git a/a b/a
```

```
old mode 100644
```

```
new mode 100755
```

5.8. Какими файлами управлять, а каких избегать

Системы управления версиями, как правило, лучше всего в управляют текстовыми файлами, которые написаны людьми, например исходный код, где файлы не меняются от одного изменения к следующему. Некоторые централизованные систем контроля версий могут сносно иметь дело и с бинарными файлами, такими как растровые изображения.

Например, команда разработчиков игр, как правило, совмещать ее исходный код и все её двоичный данные (например, данные геометрии, текстуры, карты уровней) в системе контроля версий.

Так как, как правило, невозможно объединить два противоречивых изменения в двоичном файле, централизованные систем часто предоставляют механизм блокирования файла, что позволяет пользователю сказать: «Я единственный, кто может редактировать этот файл».

По сравнению с централизованной системой, распределенная система контроля версий меняет некоторые из факторов, которыми руководствуются принимая решение, какими файлами управлять и каким образом.

Например, распределенная система контроля версий не может, по своей природе, блокировать файлы. Таким образом, нет встроенного механизма защищающего 2 людей от конфликтующих изменений в двоичном файле. Если у вас есть команда, где несколько человек могут часто редактировать бинарный файл, тогда идея использовать Mercurial или любую другую распределенную систему контроля версий для управления этими файлами не будет хорошей.

При сохранении изменений файлов, как правило Mercurial сохраняет только различия между предыдущей и нынешней версиями файла. Для большинства текстовых файлов, это очень эффективно. Тем не менее, некоторые файлы (в частности, бинарные файлы) записываются таким образом, что даже незначительные изменения в логической структуре файла отражается в изменении во многих или большинстве байтов внутри файла. Например, сжатые файлы, особенно чувствительны к этому. Если различия между каждой последующей версией файла, всегда имеют больший размер, Mercurial не сможет хранить историю изменений файла очень эффективно. Это может повлиять на локальные ресурсы хранения и количество времени, необходимое для копирования хранилища.

Чтобы получить представление о том, каким образом это может повлиять на вас на практике, предположим, вы хотите использовать Mercurial для управления документами OpenOffice. OpenOffice хранит документы на диске как сжатый ZIP архив. Изменение даже одной буквы документа в OpenOffice, и приводит к изменению почти каждого байта во всём файле, когда вы его сохраните. Теперь предположим, что файл 2 МБ в размере. Поскольку большая часть файла меняется каждый раз, когда вы сохраняете его, Mercurial придется хранить все 2 МБ файла каждый раз, когда вы фиксируете изменения, даже если с вашей точки зрения, меняется только несколько слов за раз. Частое редактирование файла, не дружественно для системы хранения Mercurial и приведёт к эффекту переполнения хранилища.

Еще хуже, если и вы, и кто-то изменяете документ OpenOffice, и не существует удобный способа объединить вашу работу. В самом деле, нет даже хорошего способа сказать, что изменилось между соответствующими ревизиями.

Есть, несколько четких рекомендаций относительно конкретных видов файлов, с которыми нужно быть очень осторожными.

- Файлы, которые являются очень большими и несжимаемыми, например, iso cd-rom образы, будет в силу огромных размеров очень медленно клонироваться по сети.
- Файлы, которые сильно меняются от одного изменения к следующему могут быть дорогостоящим для хранения, если вы их изменяете часто, и конфликты из-за одновременного изменения могут быть сопряжено с трудностями.

5.9. Резервные копии и мониторинг.

Mercurial поддерживает полную копию истории в каждом клоне, всем, кто использует Mercurial для совместной работы над проектом потенциально могут выступать в качестве источника резервных копий в случае катастрофы. Если центральное хранилище становится недоступным, можно построить просто заменить путем клонирования копии хранилища от другого разработчика и, получить любые изменения, которые, возможно, не видели другие.

Достаточно просто использовать Mercurial для оффлайновых резервных копий и удаленных зеркал. Настройка периодического задания (например, через команду *cron*) на удаленном сервере, чтобы вытаскивать изменения из вашего основного хранилища каждый час. Это будет немого сложнее в маловероятном случае, когда вы поддерживаете часто изменяемый набор мастер-репозиториев, в этом случае вам нужно сделать небольшой сценарий, чтобы обновлять список репозиториев для резервного копирования.

Если вы выполняете традиционное резервное копирование мастер-репозитория на ленту или жесткий диск, и вы хотите создать резервную копию хранилища с именем *myrepo*, *hg clone -U myrepo myrepo.bak* для создания клона *myrepo* перед началом резервного копирования. Опция `-U` не проверяет рабочий каталог после завершения клонирования, поскольку излишне и резервное копирование занимало бы больше времени.

Если вы затем используете *myrepo.bak* вместо *myrepo*, вам будет гарантирована возможность получить снимок репозитория, не боясь что какой-то разработчик вставит изменение в середине резервного копирования.

Window size: x
Viewport size: x