Системы контроля версий

Что такое управление версиями и зачем оно?

Итак, предположим, вы разрабатываете программу. Для простоты сначала предположим, что вы разрабатываете ее в одиночку и для собственного удовольствия. Как это выглядит?

Создаем директорию MyApplication, в ней - собственно файлы программы (скажем, \*.c). Далее начинается итерационный процесс: написали некоторое количество кода → скомпилировали → запустили → обнаружили ошибки или недоработки → отредактировали исходники → … Всё просто, в целом. Но через некоторое время начинаются нетривиальные вещи. Например:

* Последнее изменение сделало программу хуже, чем она была до того. Хочется вернуть всё "как было". Хорошо, если не успели закрыть редактор - делаем Undo. А если закрыли?
* Хотим приделать к программе новую функциональность, но сразу ясно, что, во-первых, сразу после написания она будет глючить (т.к. еще не отлажена), а во-вторых, в сложной программе добавление новой функциональности может добавить ошибок в ранее работавшую часть программы. Хочется, в случае чего, откатиться к заведомо работающему варианту и начать всё сначала.
* Если программа предназначена не только для вас лично, то вам придется делать "релизы", т.е. выпускать заведомо работающие версии программы. Для подготовки релиза вам придется фиксить некоторое количество багов. Это довольно скучный процесс, хочется разбавлять его добавлением новых возможностей в программу (это, как правило, интереснее). Но нельзя: добавление новых фич обязательно добавит багов, и вы никогда не доедете до релиза.

Первое, что приходит в голову для решения всех этих проблем - резервные копии. Скажем, каждую работающую версию программы упаковывать в MyApplication\_v0.001.tar.gz, в случае чего - откатываться. Для подготовки к релизу - сделать копию директории MyApplication, в которой только фиксить баги, а в основной - добавлять функционал. При этом подходе через некоторое время разработки у вас появится куча \*.tar.gz и не меньше копий рабочей директории. Очень быстро вы начнете путаться, где что.

Если вы разрабатываете программу не в одиночку, а, скажем, с группой товарищей, появляются новые неочевидности:

* Вы добавили в программу (в свою копию) возможность A, а ваш товарищ (в свою) - возможность B. Хотелось бы, чтобы у обоих была версия с обоими возможностями.
* Вы добавили в программу некоторую функцию, ваш товарищ добавил ее же, но реализовал по-другому. Какую версию выпускать и поддерживать дальше?
* Ваш товарищ в какой-то момент разработки понял, что программу можно улучшить и упростить, если изменить ее архитектуру, что-то обобщить, что-то вынести в отдельный модуль и т.п., и начал переделку (это называется умным словом рефакторинг). Вы же в это время в старой версии программы находите баги и фиксите их. Т.к. в версии вашего товарища в основном тот же код, эти баги наверняка есть и там - хотелось бы их сразу пофиксить.

Даже в случае двух разработчиков, при "ручном" согласовании изменений после определенного объема программы эти неочевидности превращаются в окончательную неразбериху. При разработке сколько-нибудь большой программы в конторе "ручное" согласование изменений видится почти невозможным.

Все эти проблемы призваны решить специальные программы - системы контроля версий (version control systems, vcs).

Общие свойства VCS

Если вы работаете с vcs, то у вас есть место, где хранится вся история изменений в программе - это место называется репозиторий (repository). Для внесения изменений в программу вам нужно получить копию текущей разрабатываемой версии - она называется вашей рабочей копией (working copy). Процесс получения рабочей копии традиционно называется словом "чекаут" (checkout). В случае коллективной разработки, вам придется поддерживать актуальность вашей рабочей копии (проверять, что там остальные наработали) - это обычно называется "апдейт" (update). Чтобы ваши изменения попали в репозиторий, вам нужно сделать коммит (commit) (еще это называется словом checkin). Часто словом commit называют собственно сам набор изменений, зафиксированный этой операцией (более точный термин - changeset). Каждое зафиксированное изменение называют ревизией. Вы можете в любой момент откатить свою рабочую копию к любой предыдущей ревизии - это называется revert.

Для того чтобы отслеживать такие вещи, как релизы, или варианты рефакторинга программы, или тестировать версии с новыми функциями - в репозитории создаются ветви разработки (branches). Для согласования хода разработки в разных ветках (она может вестись, и обычно ведется, параллельно) время от времени делают слияние веток (merge) - когда к одной ветке применяются те же изменения, что были проделаны в другой. Если какое-то место какого-то файла было по-разному изменено в разных ветках, при слиянии возникает конфликт (merge conflict) - его приходится решать вручную, выбирая один из вариантов или как-то логично их объединяя.

Так как ревизий в репозитории обычно довольно много, возникает необходимость некоторые из них как-то помечать (например, выпущенные релизы, или просто состояния программы, в которых она заведомо работала). Такие пометки называют тегами (tags). Таким образом, теги - это поименованные ревизии в репозитории.

Централизованные vs распределенные VCS

Всё множество VCS можно разделить на два класса - централизованные и распределенные.

В случае с централизованной VCS репозиторий хранится на одном сервере, и все разработчики работают с ним. Очевидное преимущество: простое управление выпуском релизов и вообще ходом развития программы, раз весь код в одном месте. Очевидный недостаток: если с сервером что-то случится, работа всех разработчиков пропадет (даже в случае регулярных бэкапов - пропадет работа всех разработчиков, скажем, за последнюю неделю). Известные примеры централизованных vcs - CVS, Subversion, Perforce.

В случае с распределенной VCS, каждый разработчик имеет полную копию репозитория и работает с ним, время от времени синхронизируясь с репозиториями остальных. Таким образом, для организации целенаправленной разработки потребуются какие-то не технические, а организационно-административные средства. Зато получаем множество преимуществ:

* Раз у каждого есть полная копия репозитория, работа всех разработчиков пропасть не может вообще.
* Часто выполняемые операции - прежде всего, commit - происходят почти мгновенно, т.к. не требуют соединения по сети.
* Каждый разработчик может создавать в своем репозитории ветки для каких-то экспериментов, всем остальным даже не нужно знать об этом.
* Т.к. в распределенных VCS предполагается регулярная синхронизация репозиториев, в них гораздо более эффективно реализована операция слияния веток (здесь это одна из базовых операций, в отличие от централизованных VCS, где это делается нечасто).
* Каждый разработчик может взять ("утянуть") у другого один или несколько коммитов, применив их к своему коду.