Введение в системы управления версиями

Отдел R&D

Маркетинговая группа Текарт

29 апреля 2009 г.

Университет Текарт

Системы управления версиями

- отслеживают изменения файлов
- сохраняют информацию об изменениях в репозитории
- рабочая копия отражает состояние файлов на определенный момент

Для разработчика

- навигация по истории разработки
- безопасное экспериментирование
- безопасная коллективная работа
- документирование всех изменений

Для менеджера проектов и клиента

- отслеживание внесенных изменений
- оценка количества и качества изменений
- привязка изменений к поставленным задачам
- формирование отчетности
- разрешение конфликтных ситуаций

Для системного администратора

- отслеживание внесенных изменений
- дополнительный источник информации если возникли проблемы
- резервная копия файлов приложения

Не только для исходного кода

- управление версиями конфигурационных файлов
- работа над документами в текстовых форматах
- средство резервного копирования
- средство синхронизации
- версионная база данных в собственных программах

Репозиторий и рабочая копия

Репозиторий – база данных, хранящая полную историю проекта. Может быть доступен как локальный каталог или сетевой ресурс.

Рабочая копия – копия файлов проекта, соответствующая его состоянию в определенный момент времени. Представляет собой локальный каталог на машине разработчика.

Рабочая копия – личное пространство разработчика, каждый разработчик работает со своей рабочей копией.

Ветви

- Альтернативные, независимые друг от друга направления разработки.
- Документы в разных ветвях имеют одинаковую историю до точки ветвления и разную – после.

Варианты ветвления

Всего несколько примеров:

- по линиям разработки (стабильная, тестируемая, экспериментальная версии)
- по версиям продукта
- по подсистемам продукта
- по задачам
- по разработчикам или отделам

Подробнее: "Streamed Lines: Branching for Parallel Development"

Основные операции

checkout	получение рабочей копии из репозитория
update/sync	обновление рабочей копии (например, проверка изменений, выполненных другими разрабочиками)
commit	запись изменений в репозиторий
branch	создание ветки
diff	поиск различий между файлами
merge	объединение изменений, например, между разными ветками

Конфликты

Один и тот же фрагмент файла изменен в разных ветках или разных рабочих копиях – конфликт.

Разрешение конфликта:

- добавить внешние изменения и заново применить свои
- заменить внешние изменения своими собственными

Если задействовано несколько разработчиков – это их общее решение.

Централизованные системы

CVS, Perforce, SVN

- единый репозиторий на главном сервере
- рабочая копия на клиентской машине
- изменения доступны всем только после записи в репозиторий

Плюсы и минусы централизованных систем

Плюсы:

- легко администрируются
- хорошо известны разработчикам
- хорошо поддерживаются инструментальными средствами

Минусы:

- работа только в online
- много сетевых операций – низкая производительность
- изменения фиксируются централизованно – невозможна независимая работа
- в случае краха сервера – резервная копия или теряем все

Arch, Bazaar, Darcs, Git, Mercurial

- нет центрального репозитория
- у каждого разработчика – свои клиентская копия и репозиторий
- разработчики могут обмениваться изменениями между своими репозиториями

Плюсы и минусы распределенных систем

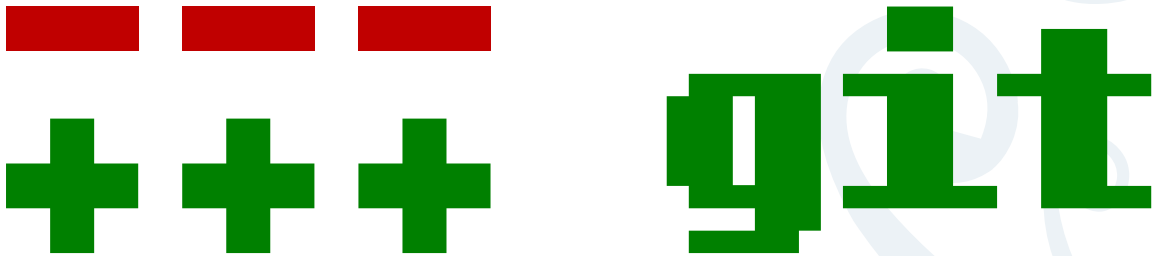
Плюсы:

- можно работать в offline
- меньше сетевых операций – выше производительность
- независимая работа и выборочный обмен изменениями
- информация дублируется, потеря репозитория не страшна
- можно реализовать аналог централизованной системы

Минусы:

- концептуально сложнее (зато гибче)
- меньшая (пока) степень поддержки коммерческими инструментальными средствами

Система управления версиями Git



http://git.scm.org

История

Сначала:

- создана Линусом Торвальдсом для разработки ядра Linux в 2005 году
- проектировалась как полная противоположность SVN
- первые версии: недружественные к пользователю, нет поддержки к Windows

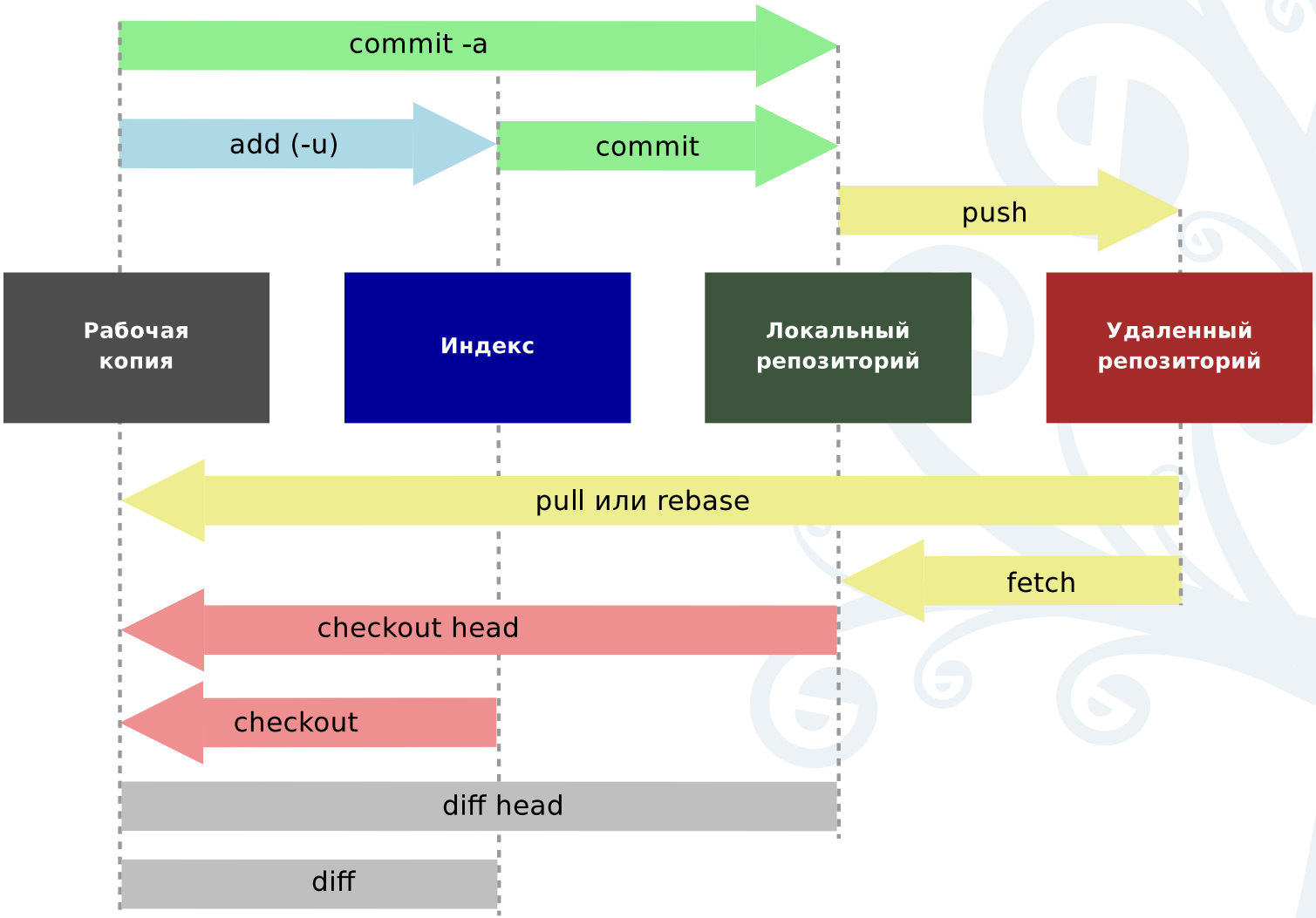
Сейчас:

- удобная система команд
- есть версия под Windows
- используется в крупных проектах: Linux, Perl, Gnome, Samba, X.Org, Qt, Android

Преимущества

- распределенная система
- эффективная поддержка веток
- быстрота работы и компактность репозитория
- универсальный сетевой доступ: http, ftp, rsync, ssh, git
- возможность интеграции с другими системами контроля версий
- различные модели совместной работы
- прозрачная архитектура, toolkit-дизайн

Схема работы



Университет Текарт

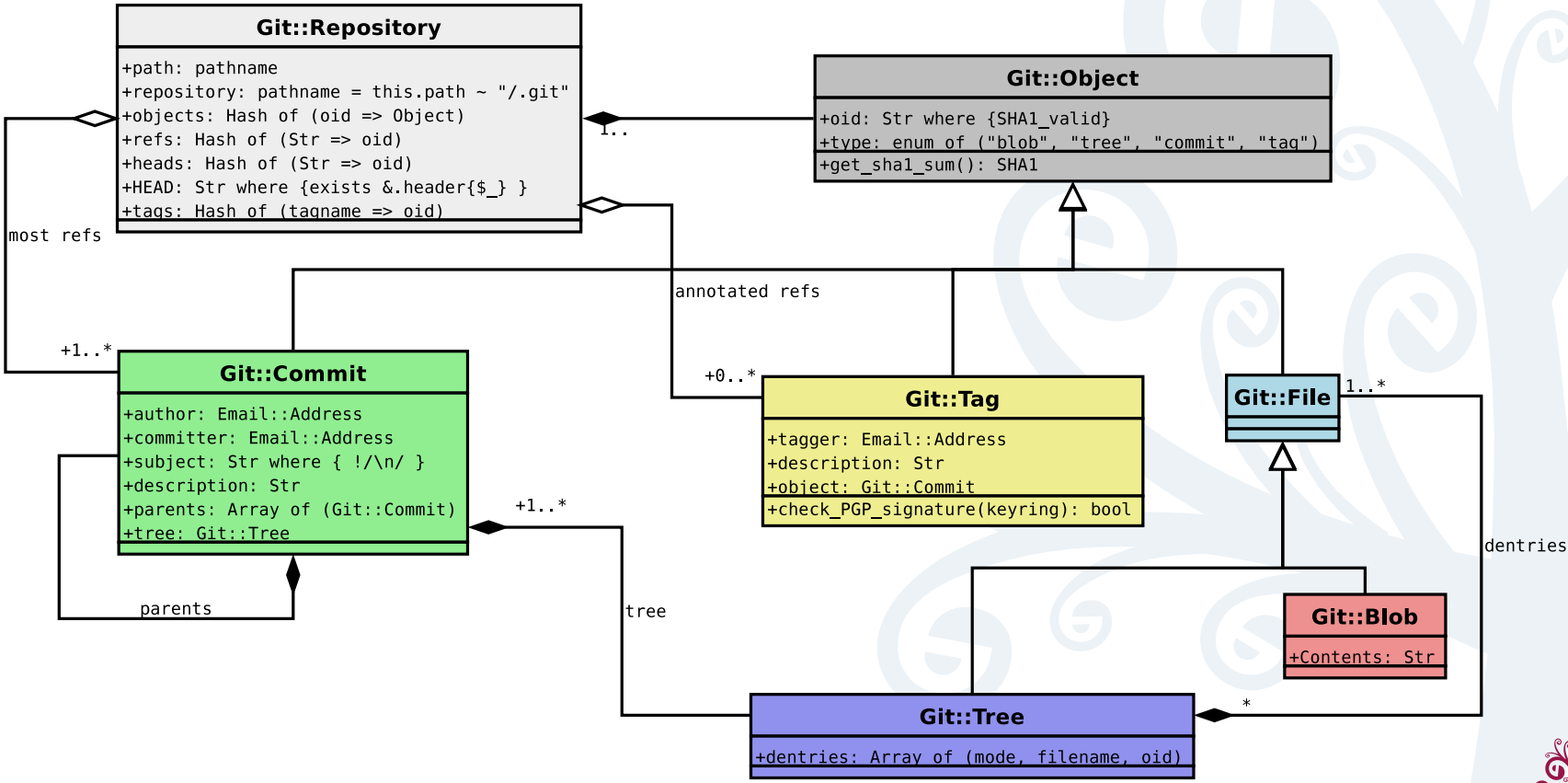
Рабочая копия

- Содержит файлы, соответствующие состоянию репозитория в определенный момент времени
- Репозиторий – в подкаталоге .git
- Можно указать список файлов, которые система должна игнорировать

Индекс

- Промежуточная область, в которой формируется содержимое коммита
- Коммит может включать в себя не полные файлы, а отдельные их фрагменты
- Коммит можно формировать интерактивно

Локальный репозиторий



Университет Текарт

Объекты репозитория

Собственно объекты: `.git/objects`

blob содержимое файла: blob + размер + контент + SHA1

tree директория проекта: другие деревья и blob'ы

commit единица истории проекта: дерево корневого каталога + родители + SHA1

Идентификатор коммита определяется его содержимым.

Ссылки: `.git/refs`

- `refs/heads/master` – ветка мастер
- `refs/tags/1.2.1` – ссылка на версию 1.2.1

Удаленный репозиторий

- основные протоколы доступа: git для чтения, ssh для чтения и записи
- репозиторий, предназначенный только для чтения и клонирования, может не иметь рабочей копии
- локальный репозиторий сохраняет ссылку на репозиторий, с которого был клонирован
- можно добавлять ссылки на другие удаленные репозитории

Toolkit Design

Основные принципы

- Каждой команде соответствует отдельная программа:
  git-status, git-add, git-commit
- Единый фронтенд git, являющийся диспетчером для вызова этих программ:
  git status, git add, git commit
- Отдельный команды не только для пользовательских, но и для низкоуровневых операций:
  git-hash-object, git-mktree, git-unpack-file

Настройка

- Пользовательские настройки

`$HOME/.gitconfig`

- Информация о пользователе

```
git config --global user.name "User Name"
git config --global user.email user@domain.ru
```

- Настройка пользовательского интерфейса

```
git config --global color.pager true
git config --global color.ui auto
```

Создание репозитория

```
$ git init
```

- выполняется в каталоге проекта
- создает каталог .git, в котором размещается репозиторий
- по умолчанию репозиторий пуст

Просмотр состояния репозитория

```
$ git status
```

staged — измененные файлы, добавленные в индекс

unstaged — измененные файлы, не добавленные в индекс

untracked — измененные файлы, не отслеживаемые в данный момент системой (например, новые файлы)

`.gitignore` – список игнорируемые системой файлов

Работа с индексом

- Добавление файла в индекс:

```
$ git add $filename
```

- Удаление файла из рабочей копии и из индекса

```
$ git rm $filename
```

- Переименование файла

```
$ git mv $oldname $newname
```

Сравнение версий

- различия между содержимым индекса и рабочей копией

```
$ git diff
```

- различия между HEAD и индексом

```
$ git diff --staged
```

- различия между индексом и рабочими файлами

```
$ git diff HEAD
```

- различия между двумя коммитами

```
$ git diff $commit1 $commit2
```

Университет Текарт

Запись изменений

- Запись изменений, добавленных в индекс

```
$ git commit
$ git commit -m "Commit message"
```

- Интерактивный выбор добавляемых изменений

```
$ git commit --interactive
```

- Если после записи обнаружили, что что-то забыли

```
$ git commit --amend
```

Не только файлы, но и отдельные измененния

- Если при выполнении нескольких задач изменялись разные части файла, можно добавить в индекс только выборочные измененные фрагменты. Таким образом, каждый результирующий коммит может соответствовать отдельной задаче.
- Для добавления отдельных фрагментов удобнее всего пользоваться интерактивным режимом.

```
$ git commit --interactive
```

Просмотр истории изменений

- Вся история

```
$ git log
```

- Поиск в интервале

```
$ git log tag..branch
$ git log -10
$ git log --since="May 1" --until="June 1"
```

- Поиск по атрибуту

```
$ git log --author=max
$ git log --grep="commit.*message.*text"
```

И самое главное...

Ветки!

Ветки и теги

- ветки и теги – по сути одно и тоже: указатели на определенный коммит
- теги никогда не меняются
- ветка автоматически переходит на следующий коммит
- создание и удаление ветки – дешевая операция

```
$ git branch $branch_name
```

Решение отдельной задачи или эксперимент

Отдельная ветка для каждой задачи или эксперимента:

```
$ git branch my-new-feature dev
$ git checkout my-new-feature
# add/commit .. add/commit .. add/commit
```

Объединение с основной веткой

Если задача выполнена:

```
$ git checkout dev
$ git merge my-new-feature
```

или

```
$ git pull . my-new-feature
```

Если нет – можно удалить ветку.

Работа с удаленными репозиториями

- Клонируем репозиторий

```
git clone git://git.rd.techart.intranet/point/
```

- Просматриваем список удаленных веток

```
$ git branch -r
```

Обновление локального репозитория

- Обновление удаленных веток без добавления изменений в локальные:

```
$ git fetch
```

- Обновление с добавлением изменений:

```
$ git pull
```

- git pull = git fetch & git rebase

Все изменения – только в локальных ветках

Для внесения изменений необходимо создать локальную ветку на базе одной из веток удаленного репозитория.

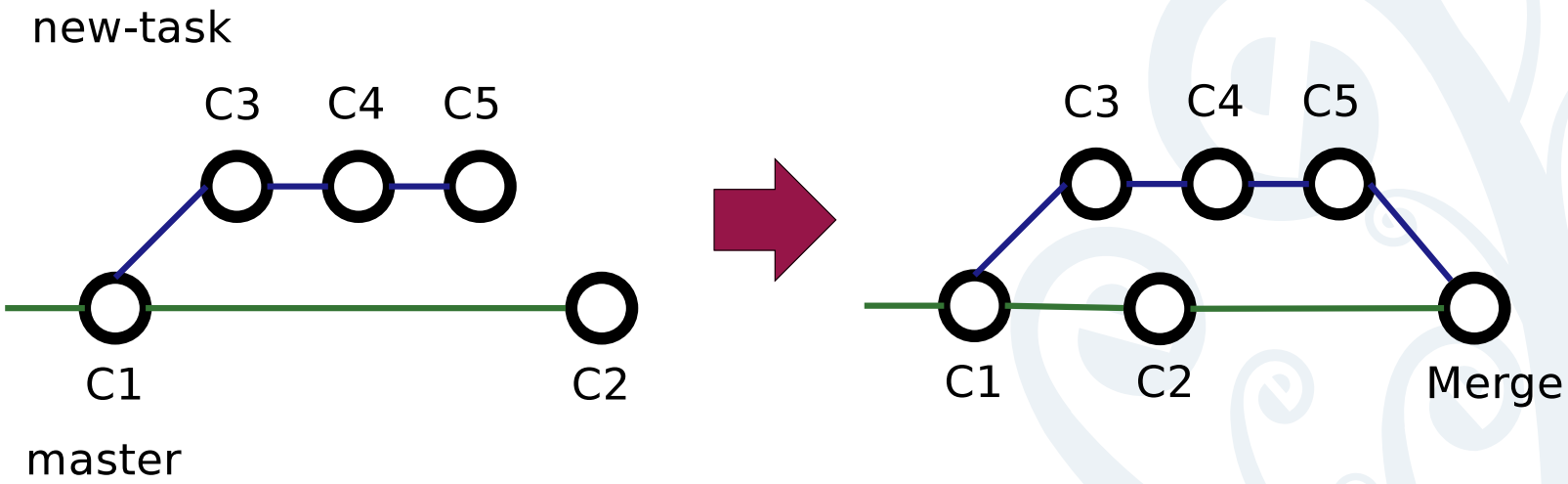
```
$ git fetch origin/master
$ git branch new-task origin/master
$ git checkout new-task
```

Публикация изменений

```
$ git push
```

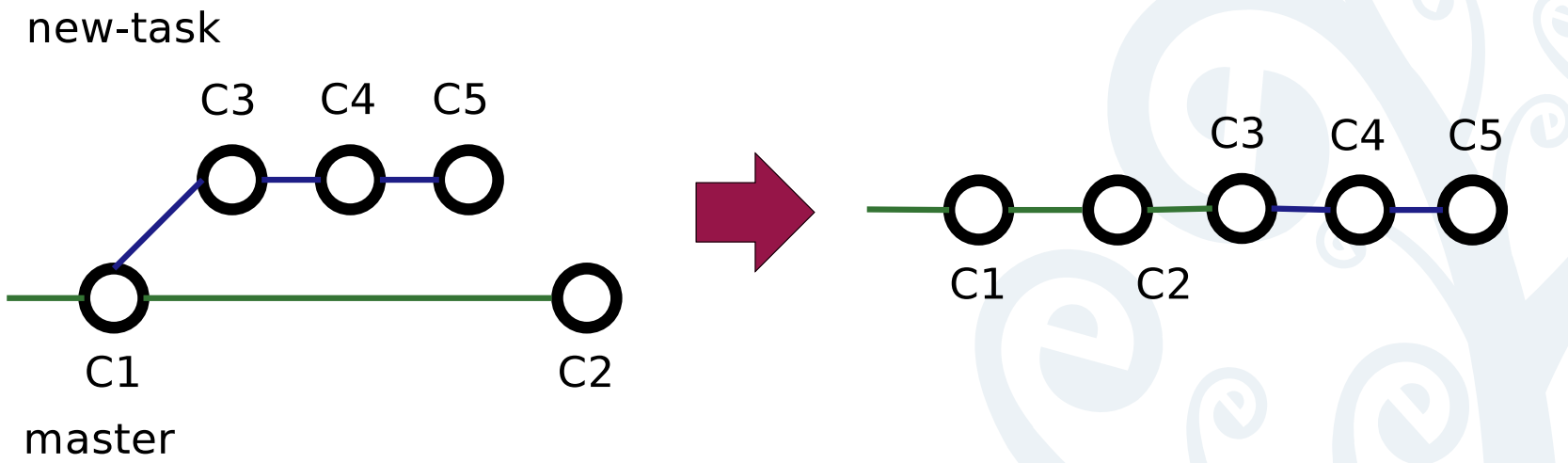
- Если с момента начала работы удаленную ветку еще никто не продвинул – fast-forward.
- Если нет – not fast-forward – ошибка
- Две стратегии: merge и rebase

Merge




```
$ git checkout master
$ git merge new-task
$ git push
```

Rebase




```
$ git fetch origin/master
$ git rebase origin/master
$ git push origin new-task:master
```

На самом деле, все сложнее, но это отдельная тема.

Университет Текарт

stash – временный буфер для сохранения изменений

- Сохраняет текущие изменения во временную ветку
- Позволяет вернуть эти изменения назад, когда понадобится

```
$ git stash
# checkout .. edit .. commit .. etc
$ git stash apply
```

Что прочитать

- `git-scm.com`
- Everyday Git With 20 Commands Or So
- Git From The Bottom Up
- Git Magic
- Git For Designers
- Git For Computer Scientists
- `gitguru.com`
- Еще немножко о Git

Продолжение следует...