СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПО

Лекция 12:

Системы контроля версий

Мотивация

- Исходный код изменяется, и изменяется «нелинейно»:
  - откат неудачных изменений
  - параллельные ветви разработки
  - командная работа
    - конфликтующие изменения
- В простейшем случае ещё можно:
  - хранить архивы всех версий
  - обмениваться кодом по почте или как-то ещё
- Но зачем? Ведь есть системы контроля версий (Version Control Systems – VCS)

Что дают VCS

- Безопасность:
  - транзакционность
  - авторизация доступа
  - легко делать backup/restore репозитория
- Организация ветвей разработки
  - ветвление истории версий
  - маркировка версий
- Эффективность
  - простота операций над историей
  - интеграция:
    - в IDE («из коробки» или в виде плагинов)
    - в OS (командная строка, интеграция в Windows Explorer)

Область

VCS хороши для:

- хранения истории правок
- текстовых форматов данных

Плохо подходят для:

- баг-трекинга, слепков баз данных, управления конфигурациями
- хранения бинарных форматов данных

Базовый сценарий использования VCS

1. Получить локальную «рабочую копию» кода из репозитория
2. Внести изменения
3. Вариативно: выполнить слияние (merge) изменений с новыми правками в репозитории
4. Зафиксировать изменения в репозитории

Виды VCS

- **Централизованные:**
  - более старый вид VCS
    - использовались ещё в 70е
- **Распределённые**
  - «новое течение»
    - первые системы – 90е, начало 2000х
    - массовое распространение – с 2005 года

Централизованные VCS

- Единое хранилище версий – центральный репозиторий
- Разработчик работает с локальной копией и отправляет изменения в центральный репозиторий
- Репозиторий виден всем (у кого есть доступ), и обмен кодом – только через него
- **Примеры**: SVN, Perforce, MS TFS, ClearCase

Распределённые VCS

- Каждый разработчик владеет копией репозитория
  - фактически, своим локальным «сервером» VCS
  - копии легко создавать: проще экспериментировать с кодом
- Передавать изменения можно между любой парой репозиториев
- Нет «главного» репозитория
  - можно строить более сложную модель «оборота» изменений
    - персональные репозитории разработчиков
    - командные и целевые репозитории (для тестирования, для ветвей разработки, автоматических сборок, и т.п.)
- **Примеры**: git, Mercurial, Bazaar

CVCS vs. DVCS

На DVCS можно всё то же, что и на CVCS

Улучшения в DVCS:

- Проще выполнять слияние ветвей
- Вся история хранится локально
  - можно работать оффлайн
  - работа в целом быстрее
- Более гибкая модель обмена изменениями
- SVN vs. git&hg:
  - svn-файлы по всему дереву каталогов,
  - git и hg хранят свои данные в отдельной папке
- Проще делать эксперименты
  - клонирование локального репозитория (в svn пришлось бы делать ветви, которые потом нельзя удалить)

CVCS vs. DVCS

Нюансы DVCS:

- разработчики привыкли к VCS, нужно перестраиваться
- у CVCS ниже «порог вхождения»
  - для работы с DVCS надо лучше понимать концепции контроля версий
- в мире CVCS есть фаворит – SVN, в DVCS пока не выявлен «победитель»

Работа с централизованным репозиторием

На примере Subversion (SVN)

Кратко про SVN

- Вышла в 2004 году как замена CVS
- Широко используется в Open source
  - Apache, GCC, Python, Ruby, Boost, …
  - репозитории на SourseForge.net, GoogleCode, etc.
- Одна из самых популярных SVC

Терминология

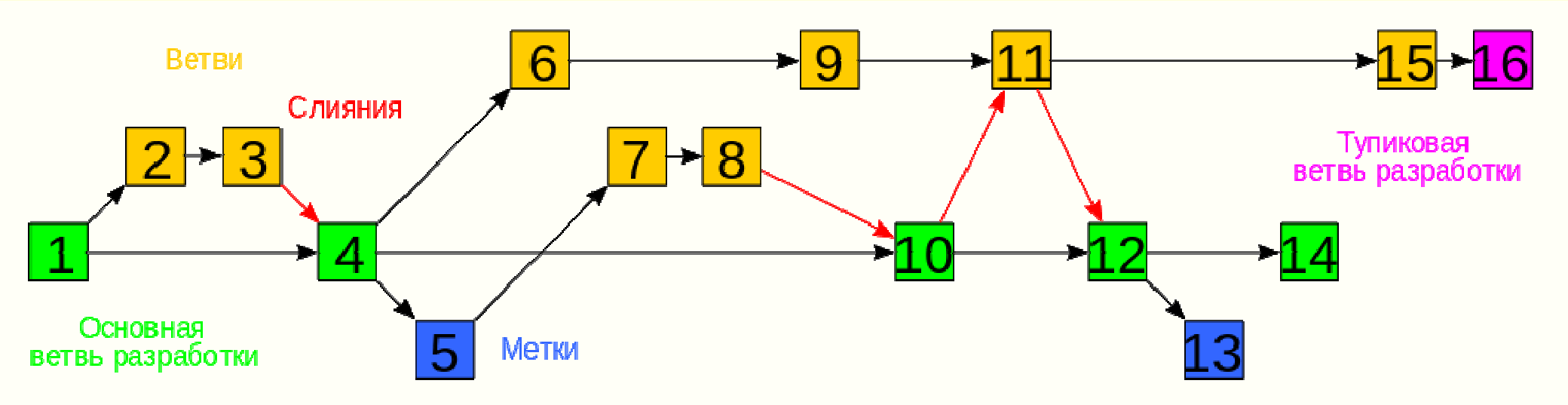
- **Репозиторий** (repository) – хранилище документов
  - место, где SVN хранит документы, историю их изменения и служебную информацию
- **Ревизия** (revision) – версия документа
  - используется автоматическая нумерация
- **Рабочая копия** (working copy) – локальная копия документов

Типовой процесс работы

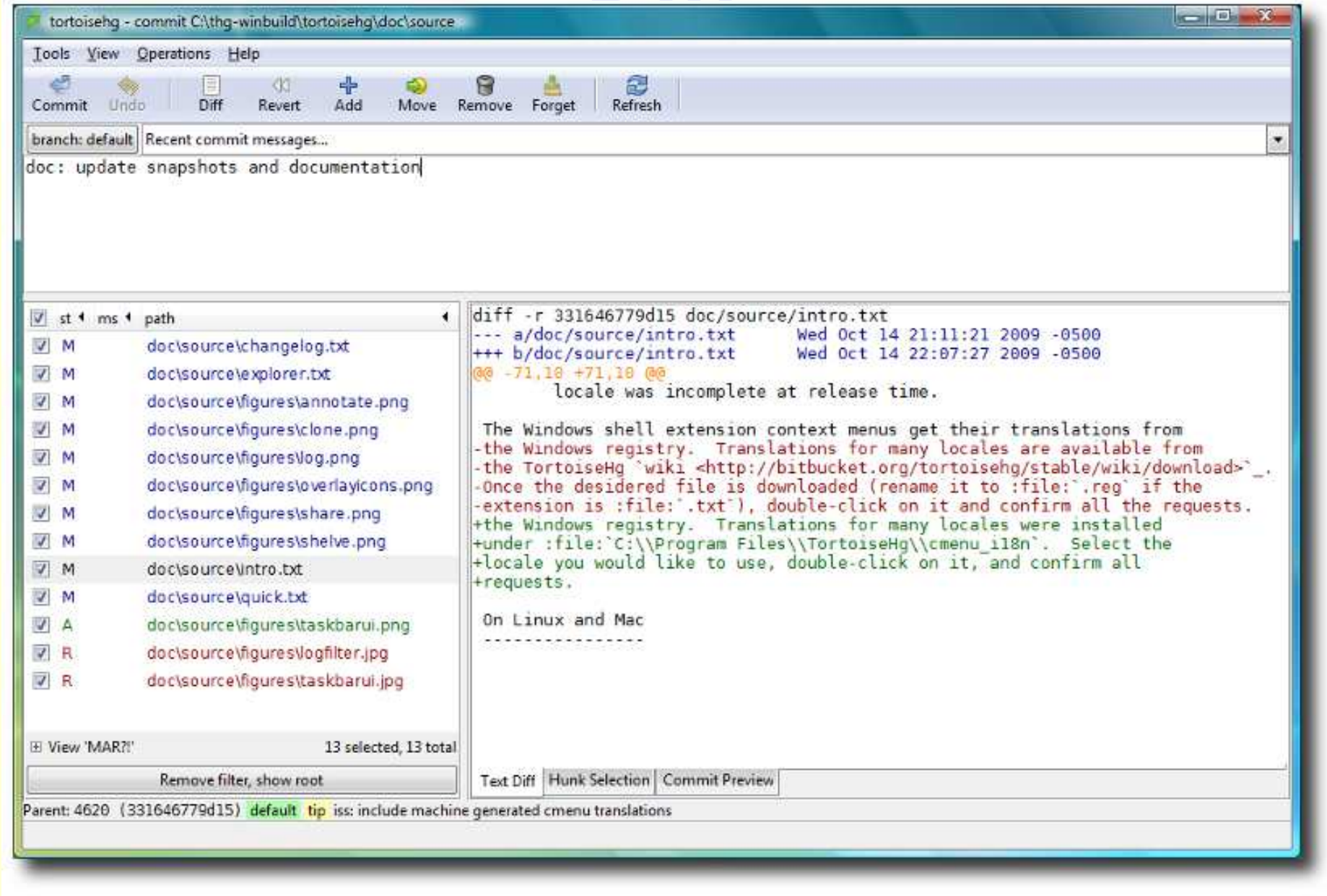
1. Получение рабочей копии:
   - svn checkout – создание (получение)
   - snv update – обновление рабочей копии до заданной ревизии
     - по умолчанию – до самой свежей
   - svn blame – информация об изменениях и их авторах
2. Изменение рабочей копии
   - svn add | mkdir | delete | move | copy
   - просмотр состояния и изменений файлов
     - svn info | status | diff
   - svn revert – откат изменений
3. По необходимости – svn update и слияние изменений
4. Фиксация изменений – svn commit

Ветвление

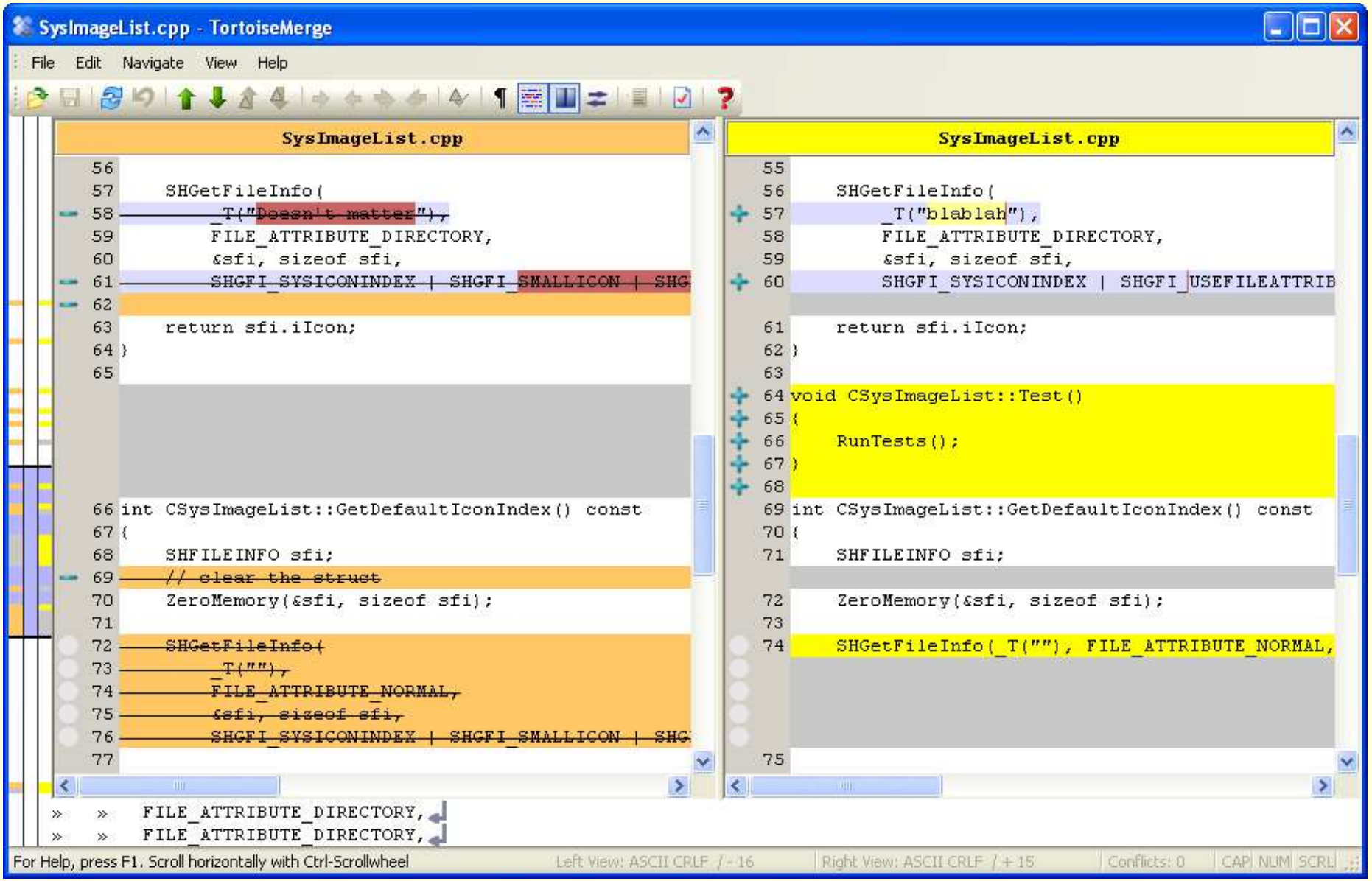
- Создание ветви – svn copy над репозиторием
- Работа с ветвями:
  - svn switch – переключение рабочей копии на другую ветвь
  - svn merge – слияние изменений одной ветви с другой ветвью



Diff


tortoisehg - commit C:\thg-winbuild\tortoisehg\doc\source
Tools View Operations Help
Commit Undo Diff Revert Add Move Remove Forget Refresh
branch: default
Recent commit messages...
doc: update snapshots and documentation
st ms path
M doc\source\changelog.txt
M doc\source\explorer.txt
M doc\source\figures\annotate.png
M doc\source\figures\clone.png
M doc\source\figures\log.png
M doc\source\figures\overlayicons.png
M doc\source\figures\share.png
M doc\source\figures\shelve.png
M doc\source\intro.txt
M doc\source\quick.txt
A doc\source\figures\taskbarui.png
R doc\source\figures\logfilter.jpg
R doc\source\figures\taskbarui.jpg
View 'MAR?!'
13 selected, 13 total
Remove filter, show root
diff -r 331646779d15 doc/source/intro.txt
--- a/doc/source/intro.txt Wed Oct 14 21:11:21 2009 -0500
+++ b/doc/source/intro.txt Wed Oct 14 22:07:27 2009 -0500
@@ -71,10 +71,10 @@
locale was incomplete at release time.
The Windows shell extension context menus get their translations from
-the Windows registry. Translations for many locales are available from
-the TortoiseHg `wiki <http://bitbucket.org/tortoisehg/stable/wiki/download>`_.
-Once the desidered file is downloaded (rename it to :file:`.reg` if the
-extension is :file:`.txt`), double-click on it and confirm all the requests.
+the Windows registry. Translations for many locales were installed
+under :file:`C:\\Program Files\\TortoiseHg\\cmenu_i18n`. Select the
+locale you would like to use, double-click on it, and confirm all
+requests.
On Linux and Mac
Text Diff
Hunk Selection
Commit Preview
Parent: 4620 (331646779d15) default tip iss: include machine generated cmenu translations

Merge


SysImageList.cpp - TortoiseMerge
File Edit Navigate View Help
SysImageList.cpp
SysImageList.cpp
SHGetFileInfo(
_T("Doesn't matter"),
FILE_ATTRIBUTE_DIRECTORY,
&sfi, sizeof sfi,
SHGFI_SYSICONINDEX | SHGFI_SMALLICON | SHG
return sfi.iIcon;
int CSysImageList::GetDefaultIconIndex() const
SHFILEINFO sfi;
// clear the struct
ZeroMemory(&sfi, sizeof sfi);
SHGetFileInfo(
_T(""),
FILE_ATTRIBUTE_NORMAL,
&sfi, sizeof sfi,
SHGFI_SYSICONINDEX | SHGFI_SMALLICON | SHG
SHGetFileInfo(
_T("blablah"),
FILE_ATTRIBUTE_DIRECTORY,
&sfi, sizeof sfi,
SHGFI_SYSICONINDEX | SHGFI_USEFILEATTRIB
return sfi.iIcon;
void CSysImageList::Test()
{
RunTests();
}
int CSysImageList::GetDefaultIconIndex() const
SHFILEINFO sfi;
ZeroMemory(&sfi, sizeof sfi);
SHGetFileInfo(_T(""), FILE_ATTRIBUTE_NORMAL,
FILE_ATTRIBUTE_DIRECTORY,
FILE_ATTRIBUTE_DIRECTORY,
For Help, press F1. Scroll horizontally with Ctrl-Scrollwheel
Left View: ASCII CRLF / - 16
Right View: ASCII CRLF / + 15
Conflicts: 0
CAP NUM SCRL

Работа с распределённым репозиторием

Mercurial (hg)

Кратко про Mercurial

- Создавалась как конкурент git
- Довольно популярная VCS:
  - Много проектов на Python/Java
  - Примеры: OpenJDK, Mozilla, NetBeans, …
- Синтаксис команд близок к SVN
- В основном, написана на Python

Рабочий процесс

1. Создание репозитория
   - hg init – «с нуля» в текущей директории
     - hg addremove – добавление имеющихся файлов в репозиторий
     - hg commit – зафиксировать начальную версию
   - hg clone – копия репозитория
2. Изменения
   - Получение изменений из удалённого репозитория: hg pull
   - Переход к ревизии: hg update
   - Состояние репозитория: hg status, hg diff, hg cat, hg log
   - Фиксация изменений: hg commit
   - Откат изменений: hg revert [--all]
3. Проталкивание изменений в удалённый репозиторий
   - просмотр отправляемых изменений: hg outgoing
   - hg push
   - при конфликтах: hg pull + hg merge + hg push

Работа с распределённым репозиторием

git

Кратко про git

- Создавался для репозитория ядра Linux
- Самая популярная DSVC
  - особенно в Open-source проектах, в Linux- и Ruby-сообществах
  - используют: Linux Kernel, GitHub (в т.ч. Ruby on Rails)

Концепция

- В зависимости от состояния, файлы хранятся:
  - Committed files – в Repository
  - Modified (редактируемые) – в Working Directory
  - Staged (отмеченные для включения в коммит) – в Staging Area
- Хранит «снимки» версий файлов
  - а не изменения между версиями

Рабочий процесс

1. Создание репозитория
   - git init – «с нуля» в текущей директории
     - git add . – добавление имеющихся файлов в Staged
   - git clone – копия репозитория
2. Изменение
   - Новая ветка:
     - git branch my-branch - создать
     - git checkout my-branch – переключиться на ветку
   - Индексация изменений в ветке: git add | rm
   - Состояние проекта (файлов): git status
   - Фиксация изменений: git commit
   - Откат изменений:
     - “unstage” - git reset
     - и откат к рабочей версии – git checkout
3. Слияние ветвей
   1. переключение на «базовую» ветвь: git checkout master-branch
   2. получение последних изменений: git pull
   3. слияние: git merge my-branch
4. Проталкивание изменений в удалённый репозиторий
   - git push

Ссылки

- SVN:
  - http://ru.wikipedia.org/wiki/Subversion
  - Офсайт: http://subversion.apache.org/
  - Клиент TortoiseSVN: http://tortoisesvn.net/
  - Краткая инструкция: http://www.source-team.com/svnfordummies
  - Ещё инструкция: http://habrahabr.ru/post/150799/
- Mercurial (Hg):
  - Офсайт: http://mercurial.selenic.com/ - там же можно получить версию вместе с GUI-клиентом TortoiseHg
  - Большой набор материалов: http://mercurial.ru/
  - Руководство по Hg от Joel Spolsky: http://hginit.com/
  - Цикл статей - переводов "Hg Init" на русский: часть 1, часть2, 3, 4, 5, 6 (если не знакомы с SVN, можно начать с ч.2)
  - Работа с TortoiseHg: http://dreamhelg.ru/2009/02/tortoisehg/
- Git:
  - Офсайт: http://git-scm.com/
  - Документация на русском: http://git-scm.com/book/ru/
  - Краткое введение: http://www.rsdn.ru/article/tools/Git.xml