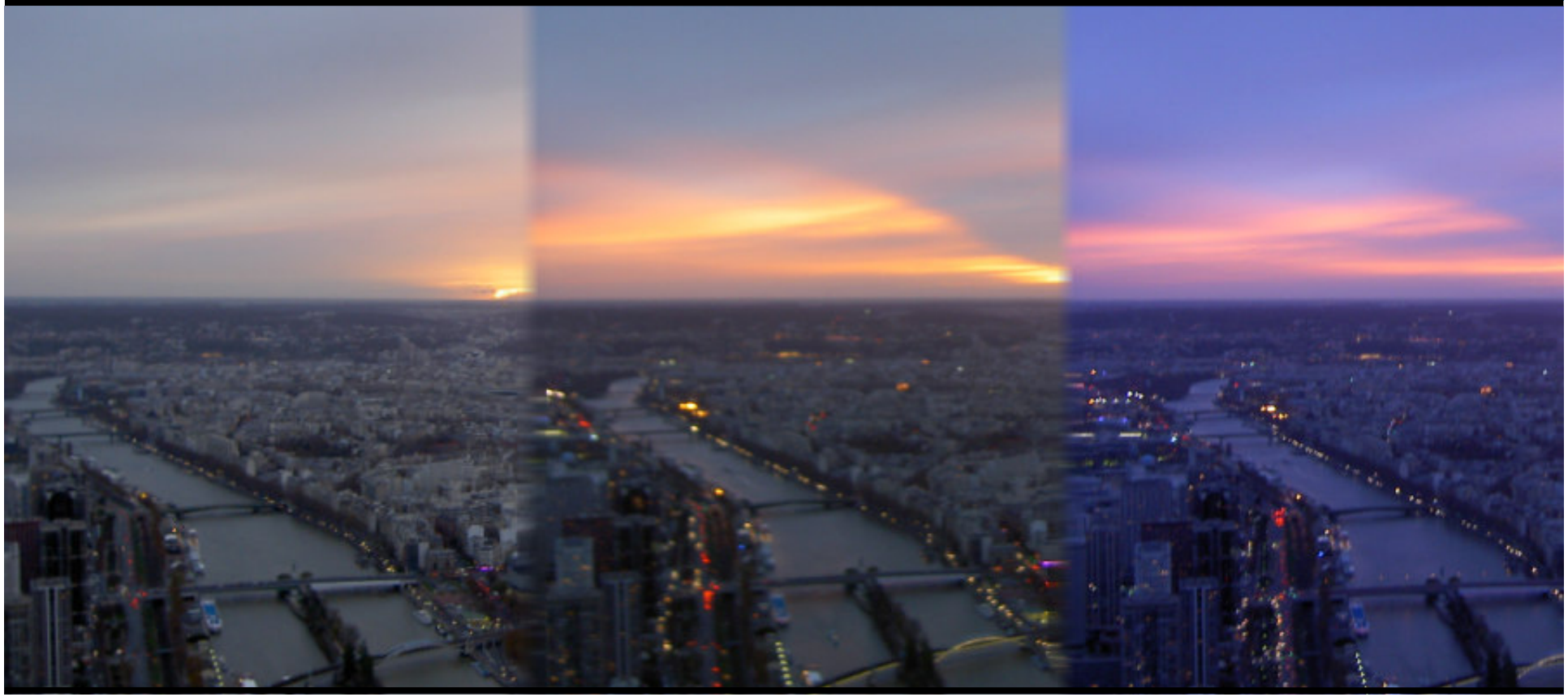
Version Control With Subversion



Jonathan Worthington

Scarborough Linux User Group

Happy New Year!

My new year's resolution: prepare my talks well in advance!

I went to EuroDisney for New Years.

Version Control With Subversion

I got back last night…and remembered something…

Version Control With Subversion

Tonight's topic...

Not about how much Perl rules.

(You know that already.)

Not about virtual machines.

(You like them. They're your friends.)

Version Control

(Sounds boring, huh?)

What Does Version Control Do?

- Enables one or more people to work on the same bunch of files (for example, the source code for a program)…
  - **Safely** – if Mickey makes a change to a file, then Pluto (who doesn't know about the latest change) makes an incompatible change, the VC system will flag this up and require that the conflict is resolved.

What Does Version Control Do?

- Enables one or more people to work on the same bunch of files (for example, the source code for a program)…
  - **Securely** – if Goofy is given the ability to make changes to the files but either accidentally or maliciously makes a mess of them, it is possible to go back to an earlier version of the files to undo the damage

What Does Version Control Do?

- Enables one or more people to work on the same bunch of files (for example, the source code for a program)…
  - **Visibly** – if Minnie wants to see what Mickey, Pluto and Goofy have been doing, she can review all the changes that have been made to the files

For Little And Large

- Every large software project will use version control
- Open source projects do, and they usually make the files and change logs publicly accessible
- However, also good if there is only you developing; get a full history of what you did and the ability to roll back changes that were problematic

Which Version Control System?

- Tens, maybe hundreds of version control systems exist
- Some are proprietary, some are open source
- Some work only in the shell, some are GUI based, some provide you with a choice of both
- A few different philosophies…

Exclusive vs. Concurrent

- **Exclusive Locking**
  - Only one person can ever be editing a file at a time
- **Concurrent Development**
  - Allow multiple people to change the same file at the same time
  - Merge compatible changes, make the user deal with incompatible ones

File vs. Global Version Number

- **Individual File Versioning**
  - Every file has a version number of its own
  - A change that touches many files increments each file's version number
- **Global Versioning**
  - One version number for all files; a set of related changes increments it

Subversion

Subversion (aka SVN)

- Open Source
- Currently popular – many projects moved from CVS to SVN
- Runs on Linux, the BSDs, Windows, OSX, your dog…
- Concurrent versioning (like CVS)
- Global version number (CVS was version number per file)

Terminology

- **Repository**
  - The place where the latest copy of the files along with all of their history is stored
- **Revision**
  - A particular version of the files or a particular file

Terminology

- **Check Out**
  - Get a copy of all the files in the repository and store them on your computer
- **Update**
  - Update the copy of the files on your computer to the latest revision in the repository

Terminology

- **Check In / Commit**
  - Put the latest changes that you have made to your local copy into the repository
  - Tries to automatically merge changes if needed and if it's safe to do so
  - Check in is usually restricted to those who are trusted

svn

- The command line client for Subversion
- Check out files using the **co** command; first argument is the URL of the repository, the second is the folder to place our local copy in

```
svn co https://svn.perl.org/parrot/trunk
parrot
```

- Update using the **up** command

```
svn up
```

svn

- If we make changes, we commit them using the **`ci`** command (check in).
- It's good practice to specify a message explaining your change – make it descriptive
- Try and commit one particular change or related set of changes at a time

```
svn ci –m "Fixed the DrinkBeer method to not crash if whisky is passed instead."
```

svn

- If you want to add or remove a file, use the **`add`** and **`rm`** commands
- To rename or move a file use the **`mv`** command
- Note that you must always commit after making these changes to make them in the repository

```
svn add Whisky.pm
svn ci -m "Implement whisky drinking."
```

Creating Your Own Repository

svnadmin and svnserve

- **svnadmin**
  - Command line tool for creating and administrating repositories
- **svnserve**
  - Server that speaks the SVN protocol
  - You can also run SVN over HTTP, but we won't cover that today.

(Because I don't know how you do it, and can't be bothered to look it up, and hey, I ain't doing everything for you. Unless you pay me. ☺)

Creating Your Repository

- I tend to create an **`svn`** user and run **`svnserve`** as that user, placing the repository files in the home directory
- Here's how we create a repository called **`example`**

```
[jnthn ~]$ su -l svn
Password: **************
[svn ~]$ svnadmin create /home/svn/example
```

Access Control

- By default a repository has anonymous read and authenticated write access
- To change this, edit the configuration file

```
[svn ~]$ vi example/conf/svnserve.conf
```

- Change **anon-access** and **auth-access**

```
anon-access = none    # default was read
auth-access = write   # allows read too :-)
```

Authentication

- The simplest way is to have a password file
- In **`svnserve.conf`**, uncomment the **`password-db`** line

```
password-db = passwd
```

- Then edit the **`conf/passwd`** file

```
[users]                              # realm
mickey = iluvminnieohsomadly # user = pass
minnie = mickeyblowsmymind   # user = pass
```

Start The Server!

- Use the **`-d`** switch to run it in daemon mode (so it lives on when we **`exit`**).
- Use the **`-r`** switch to specify the root of the repositories (in my setup, the **`svn`** home directory)

```
[svn ~]$ svnserve -d -r /home/svn
[svn@jnthn ~]$ ps -e | grep [s]vnserve
22304 ?        00:00:00 svnserve
[svn ~]$ exit
```

Set Up The Repository

- We need to create an initial directory structure for our repository
- Good idea to have a **`trunk`** directory that you put everything in (for reasons beyond tonight's talk)

```
[jnthn ~]$ mkdir import
[jnthn ~]$ mkdir import/trunk
[jnthn ~]$ mkdir import/trunk/src
[jnthn ~]$ mkdir import/trunk/docs
[jnthn ~]$ vi import/trunk/README
```

Set Up The Repository

- Then use the **`import`** command to add these files as the first revision

```
svn import import/* svn://localhost:/example
 -m "Initial import."
```

- It will request the username and password (if it gets the user wrong first time, just press enter at the password prompt and it will prompt you for the user).

Use The Repository

- You don't need your initial structure any more – note it is not a working copy!

```
[jnthn ~]$ rm -rf import
```

- You need to check out a working copy

```
[jnthn ~]$ svn co svn://localhost:/example
 example
A    example/src
A    example/docs
Checked out revision 1.
```

Use The Repository

- We can now add files to the repository and check them in

```
[jnthn ~]$ cd example
[jnthn example]$ echo 'Hello, world!' > hi
[jnthn example]$ svn add hi
A         hi
[jnthn example]$ svn ci -m "Hello message."
Adding           hi
Transmitting file data .
Committed revision 2.
```

…and they all lived happily ever after.

The End.

Questions?