Eclipse, Python, Git, and Vim

Oh My!

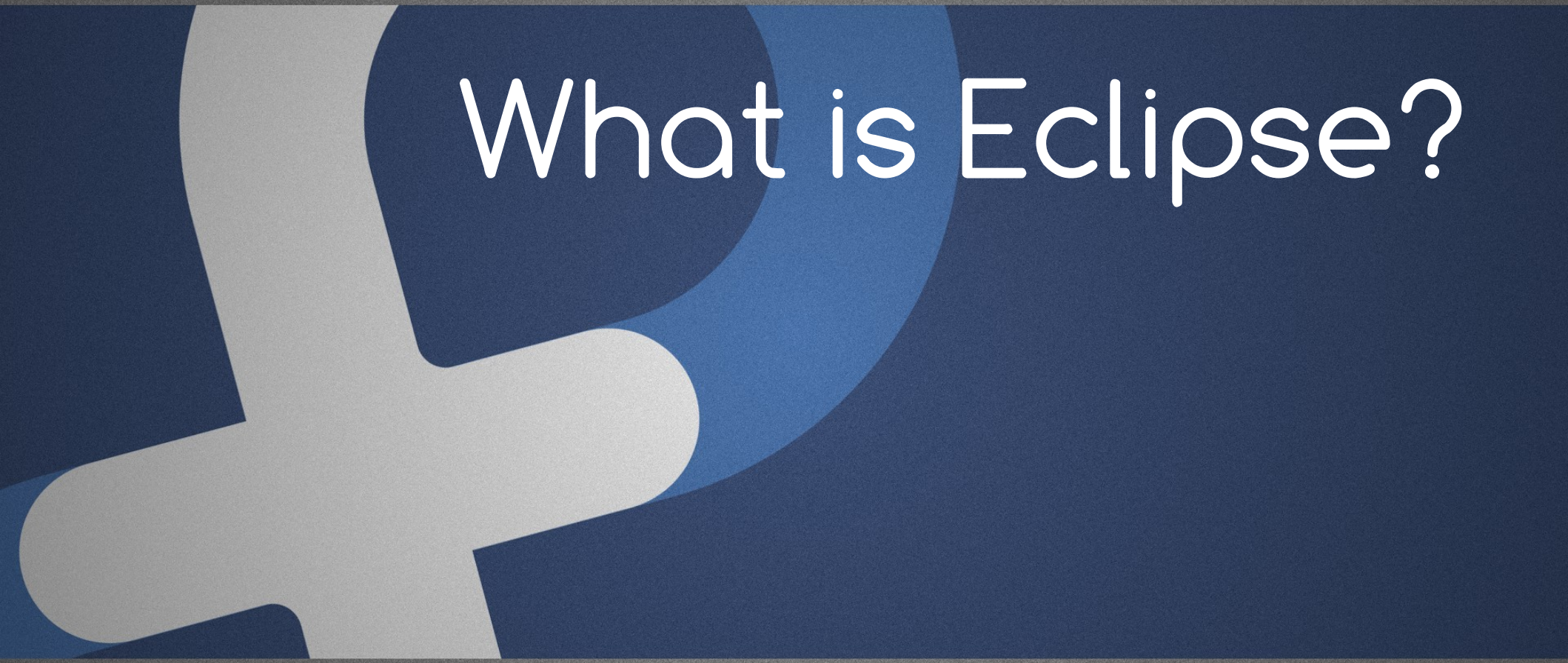
PRESENTED BY:

Jesse Keating

Senior Software Engineer, Red Hat, Inc.

Today's Topics

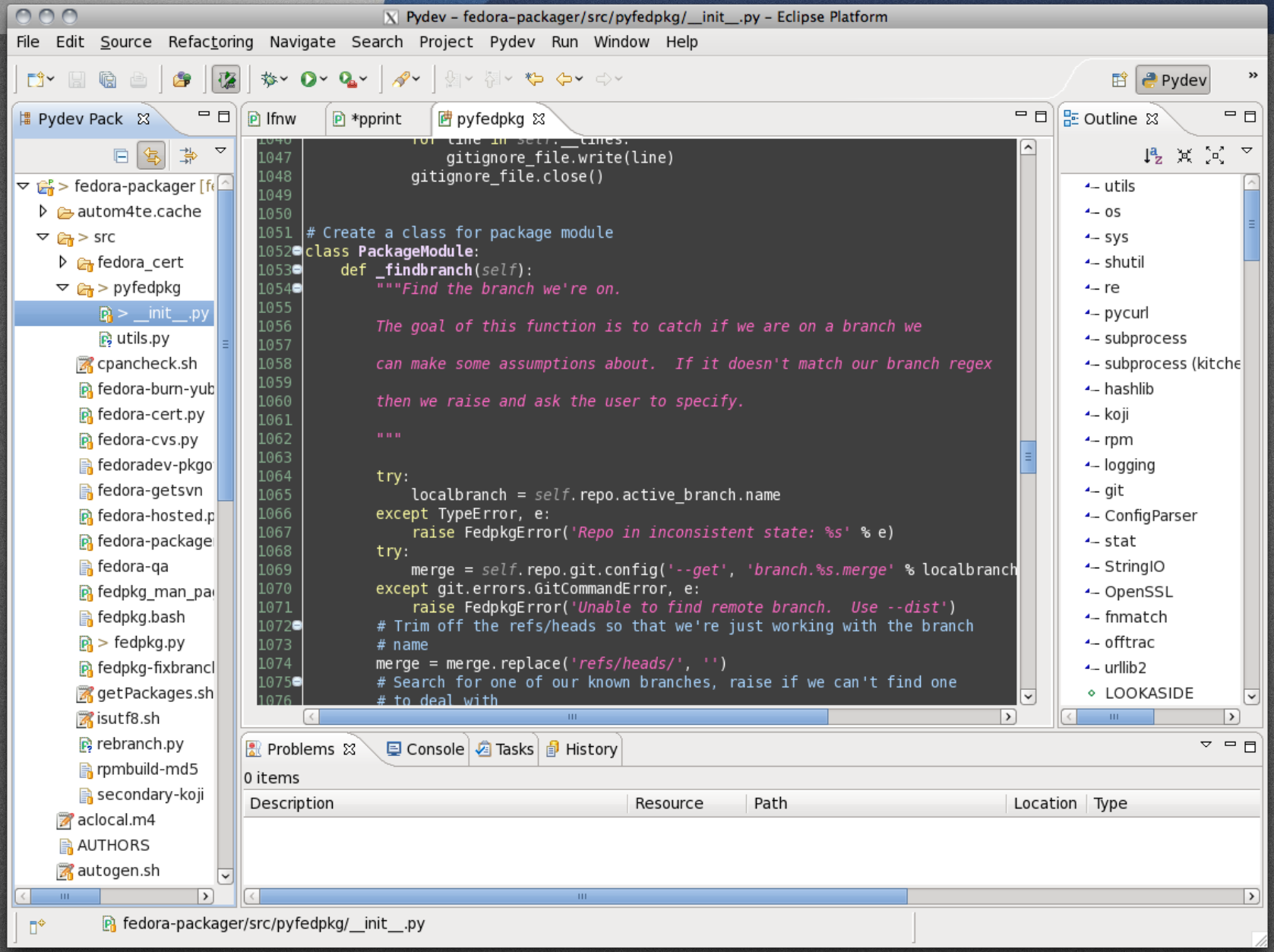
- What is Eclipse?
- Developing Python in Eclipse
- Interacting with git in Eclipse
- Using Vim with Eclipse


What is Eclipse?

Eclipse is...

- not a cheesy vampire book
- not a Japanese sports car
- not a pack of gum
- an Integrated Development Environment
- (eclipse.org is much bigger than just the IDE)

A quick tour


Pydev – fedora-packager/src/pyfedpkg/__init__.py – Eclipse Platform
File Edit Source Refactoring Navigate Search Project Pydev Run Window Help
Pydev
Pydev Pack
fedora-packager
autom4te.cache
src
fedora_cert
pyfedpkg
__init__.py
utils.py
cpancheck.sh
fedora-burn-yub
fedora-cert.py
fedora-cvs.py
fedoradev-pkgo
fedora-getsvn
fedora-hosted.p
fedora-package
fedora-qa
fedpkg_man_pa
fedpkg.bash
fedpkg.py
fedpkg-fixbrancl
getPackages.sh
isutf8.sh
rebranch.py
rpmbuild-md5
secondary-koji
aclocal.m4
AUTHORS
autogen.sh
lfnw
*pprint
pyfedpkg
gitignore_file.write(line)
gitignore_file.close()
# Create a class for package module
class PackageModule:
def _findbranch(self):
"""Find the branch we're on.
The goal of this function is to catch if we are on a branch we
can make some assumptions about. If it doesn't match our branch regex
then we raise and ask the user to specify.
"""
try:
localbranch = self.repo.active_branch.name
except TypeError, e:
raise FedpkgError('Repo in inconsistent state: %s' % e)
try:
merge = self.repo.git.config('--get', 'branch.%s.merge' % localbranch
except git.errors.GitCommandError, e:
raise FedpkgError('Unable to find remote branch. Use --dist')
# Trim off the refs/heads so that we're just working with the branch
# name
merge = merge.replace('refs/heads/', '')
# Search for one of our known branches, raise if we can't find one
# to deal with
Outline
utils
os
sys
shutil
re
pycurl
subprocess
subprocess (kitche
hashlib
koji
rpm
logging
git
ConfigParser
stat
StringIO
OpenSSL
fnmatch
offtrac
urllib2
LOOKASIDE
Problems
Console
Tasks
History
0 items
Description
Resource
Path
Location
Type
fedora-packager/src/pyfedpkg/__init__.py

Features

- Editor
- Multiple Perspectives
- Execution Testing
- Debugging
- Team (source control) Interaction
- Plugins to add lots more!

Editor

- Multiple tabs
- Language colors
- Code Completion
- Whitespace management
- (near) Real Time error checking
- Code folding/collapsing
- Spell checking
- Much more

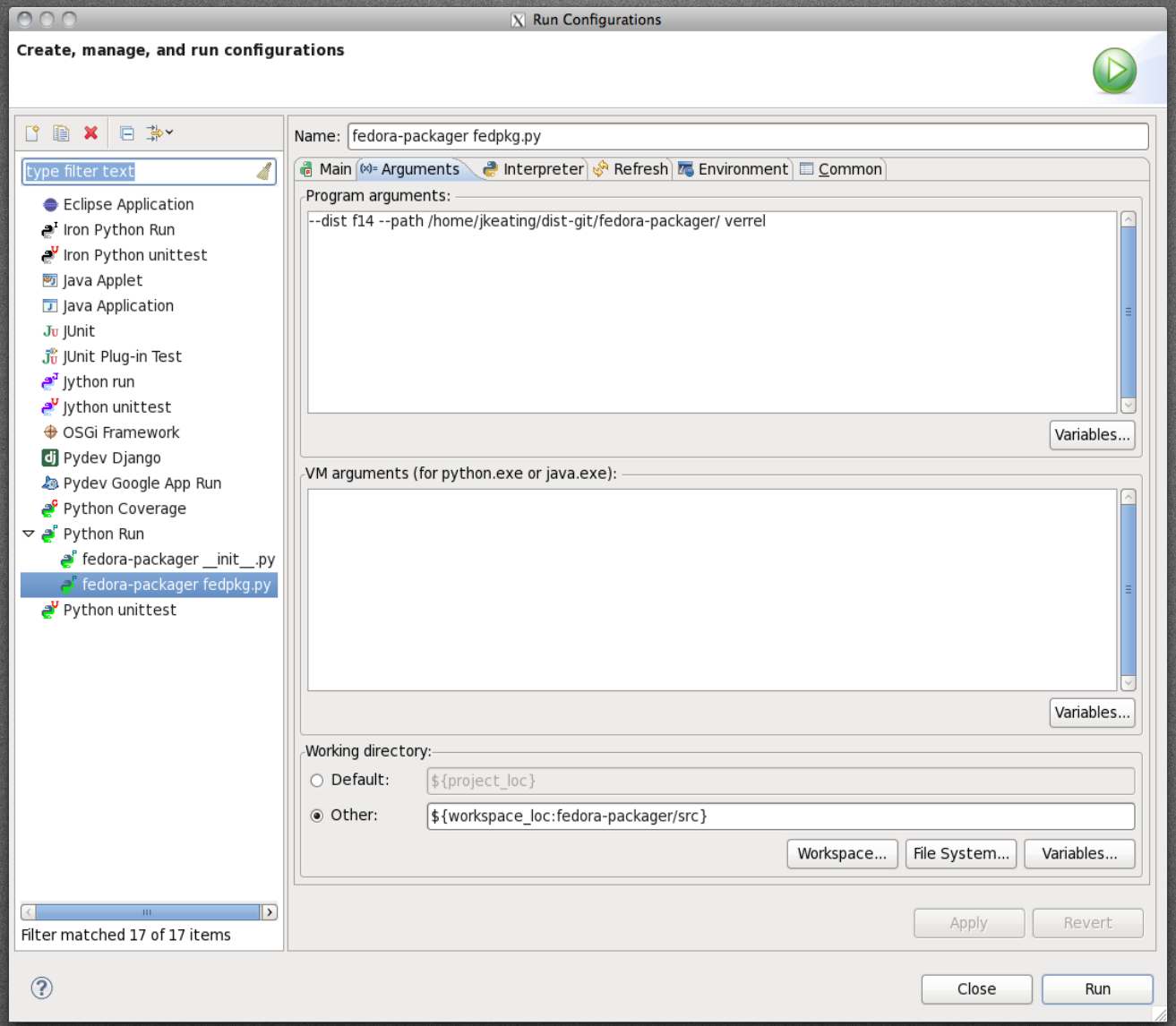
Pydev Perspective Views

- Navigation and information
  - Project explorer
  - Source file outline
  - Errors
  - Console
  - History
- Can be on their own or stacked
- Can minimize or maximize

Execution Testing

- Multiple configurations
- Custom app / interpreter arguments
- Console output
- Support for code coverage
- Support for Google App Run
- More with plugins

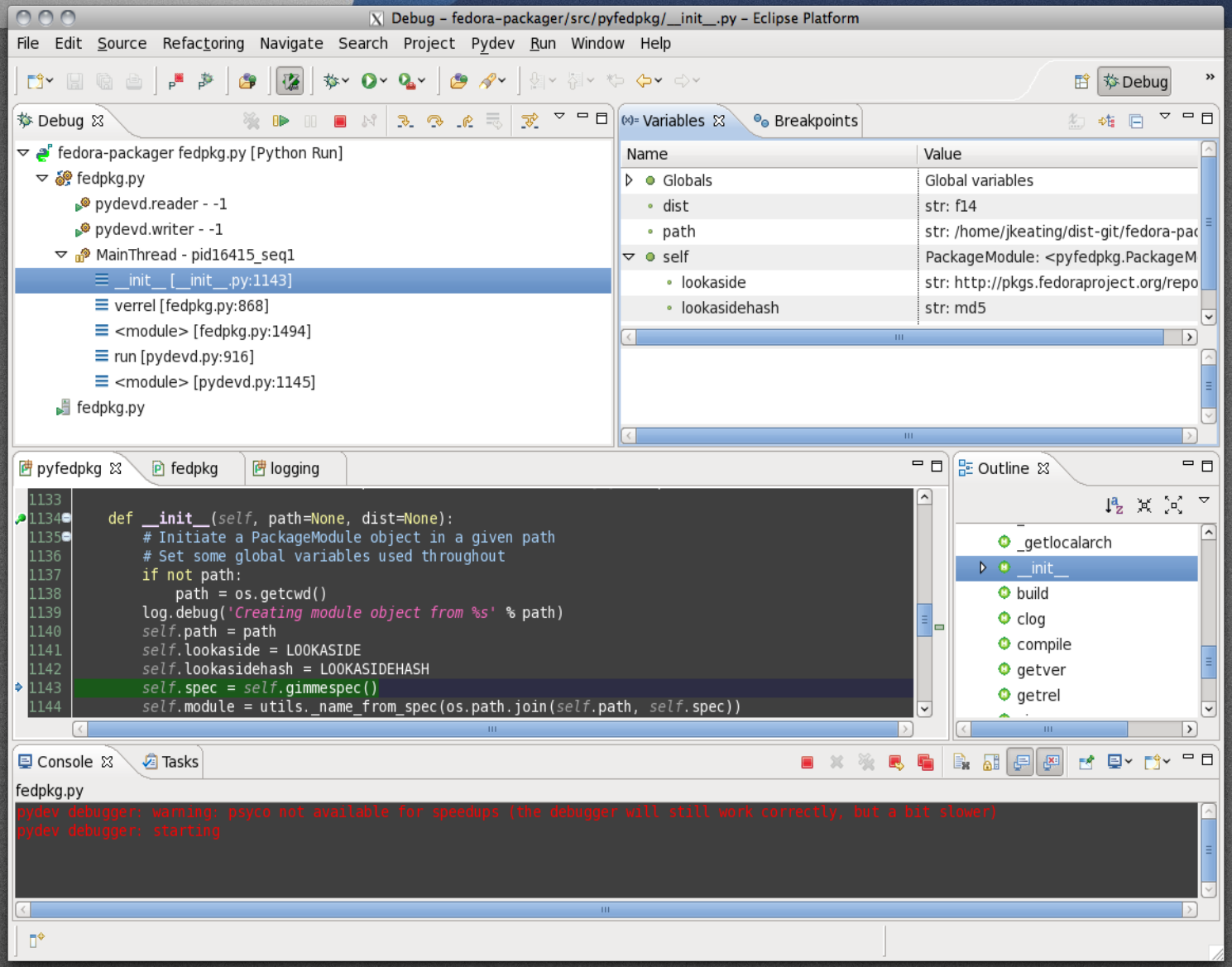
Execution Testing



Debug Perspective

- Set breakpoints
- Inspect stack data
- Step into, over, return
- Manually pause, resume
- Multiple configurations (linked with run)
- More with plugins

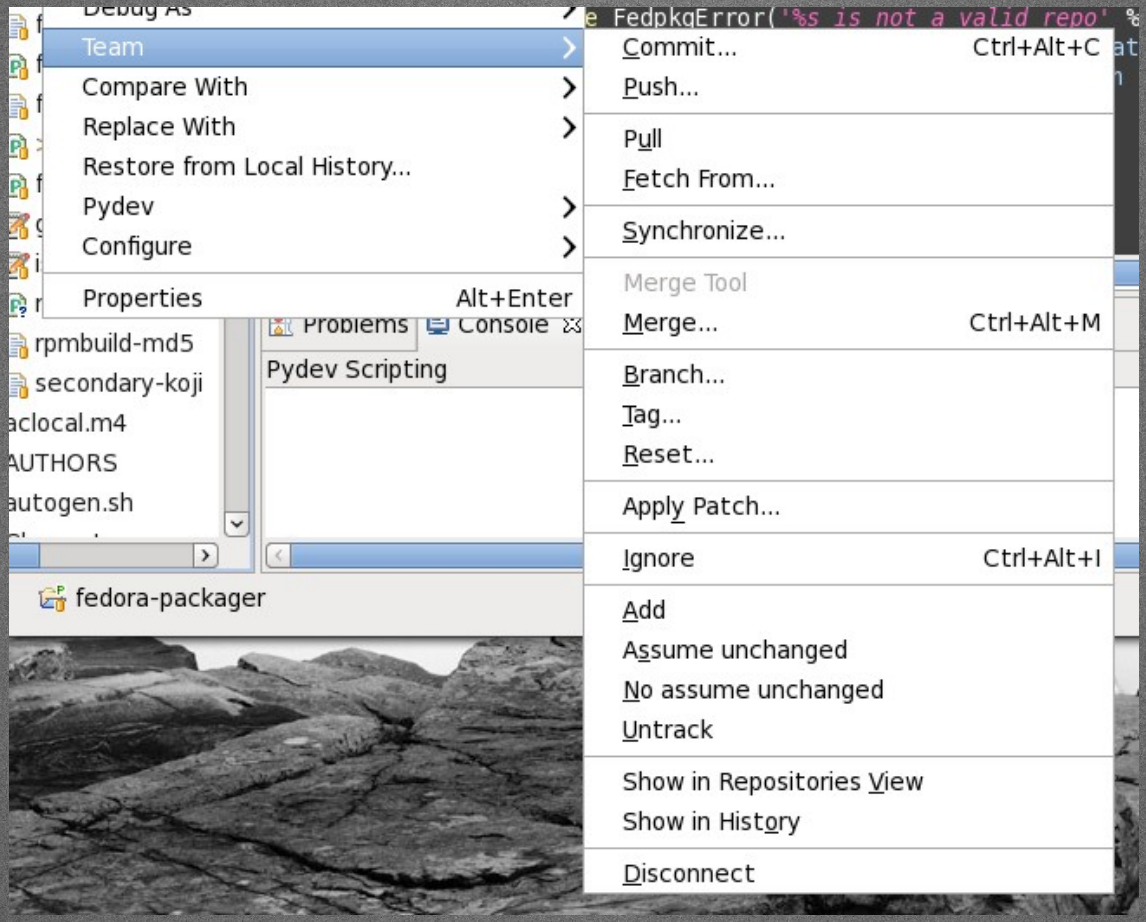
Debug Perspective



Team Controls

- Interact with source control
  - commit
  - push
  - merge
  - tag
  - More...
- Support for a variety of SCMs (with plugins)

Team Controls




Developing Python

Create a new Project

- Create a pydev project
- Create a new python package within the project
- Create a new module within the package
- Create a the script


Sling some Code

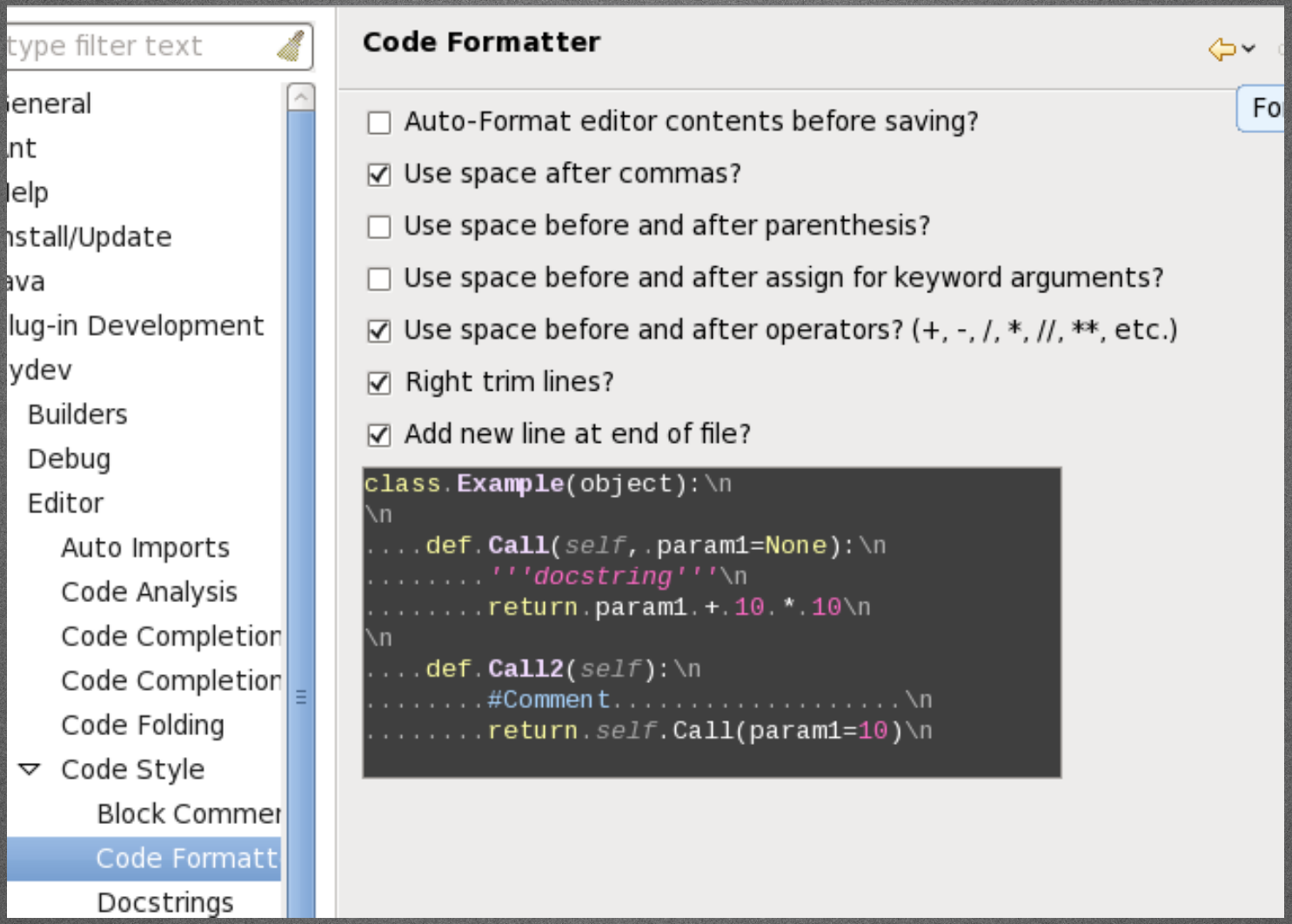
Setup a run

- Make sure script is the active tab
- “Console” view tab will automatically focus when output happens

Setup a debug

- Breakpoints are vitally important
  - Cannot be on a blank line (lost lots of time to this one...)
  - Do not have to save the file after adding a break point
- Debug perspective will automatically launch as soon as a breakpoint is encountered
- Can use console to evaluate statements

Code Formatting



Diffing

- Can diff against local history
- Could diff against previous SCM commits
- Can revert all or parts

Refactoring

- Rename items
- Create new methods from existing code
- Inline / extract a variable

Interactive Console

- Use a fresh python prompt
- Send selected code to the console
- Get execfile sent to console to continue playing with symbols

Code Testing

- Support for code unittests
  - Pydev test runner
  - Nose
  - py.test
- Support for code coverage
- Support for pylint


Interacting with git

git Interaction

- Can create new repo from existing project
- Can create new project from existing repo
- Can link existing project to existing repo

Create a git repo from project

- Share Project
- Choose git
- Create a new repository
- Profit!

Commit files to git

- No files exist in the repo by default, they have to be added/committed

Using git to aid development

- Create branches for topic work
- Diffing / committing
- Creating patches
- Resetting work
- History and repository viewing
- Merging
- Tagging


Using Vim

vrapper

- Wraps the current editor with vim like keybindings, rather than embedding vim itself
- Easy to turn on/off without restarting eclipse
- Still has command/insert modes
- Not all commands or key sequences work though, and a few bugs.

Some quick vrapper features

- Navigation (arrows or k,j,h,l)
- Searching (/,?,n,N)
- Change {word,line,etc} (c{w,$,G,gg})
- Undo / redo (u,R)
- Repeat (.)
- Yank / paste (y{...},{p,P})
- Visual mode (v)
- Command mode (:)

Some quick vrapper features

- Config file (.vrapperrc)
- Macros (q[a-z])
- Marks (m[a-z])

What's missing?

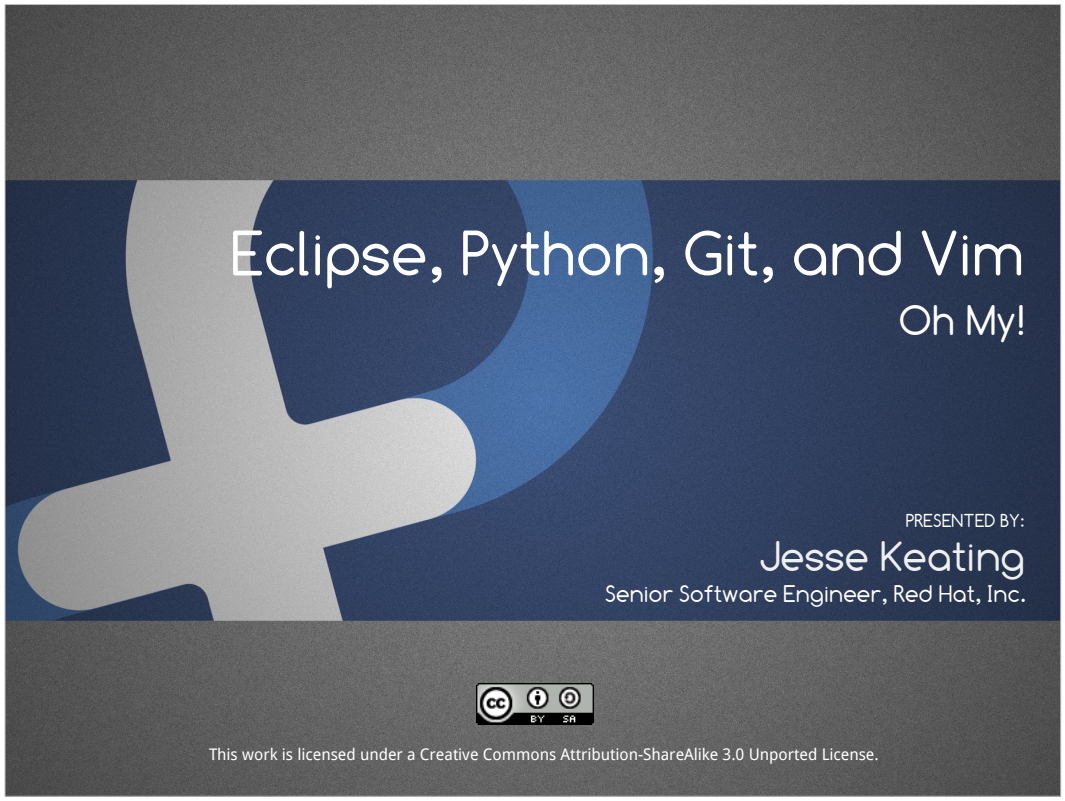
- Search and replace
- Regex searching
- Vim plugins

Summary

- Eclipse is a useful IDE
- Developing python in Eclipse is awesome
- Using git within Eclipse is handy
- Using vim within Eclipse is a godsend!

Questions?

CONTACT:
jkeating@redhat.com

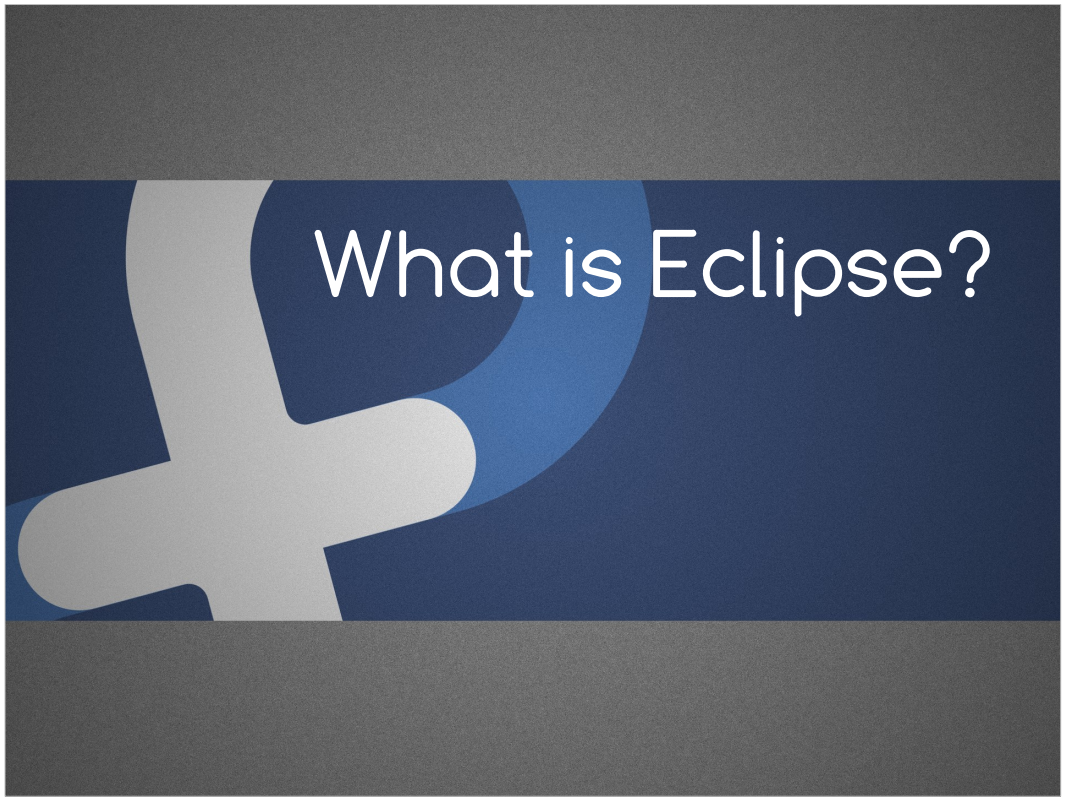


Who are you and what am I?

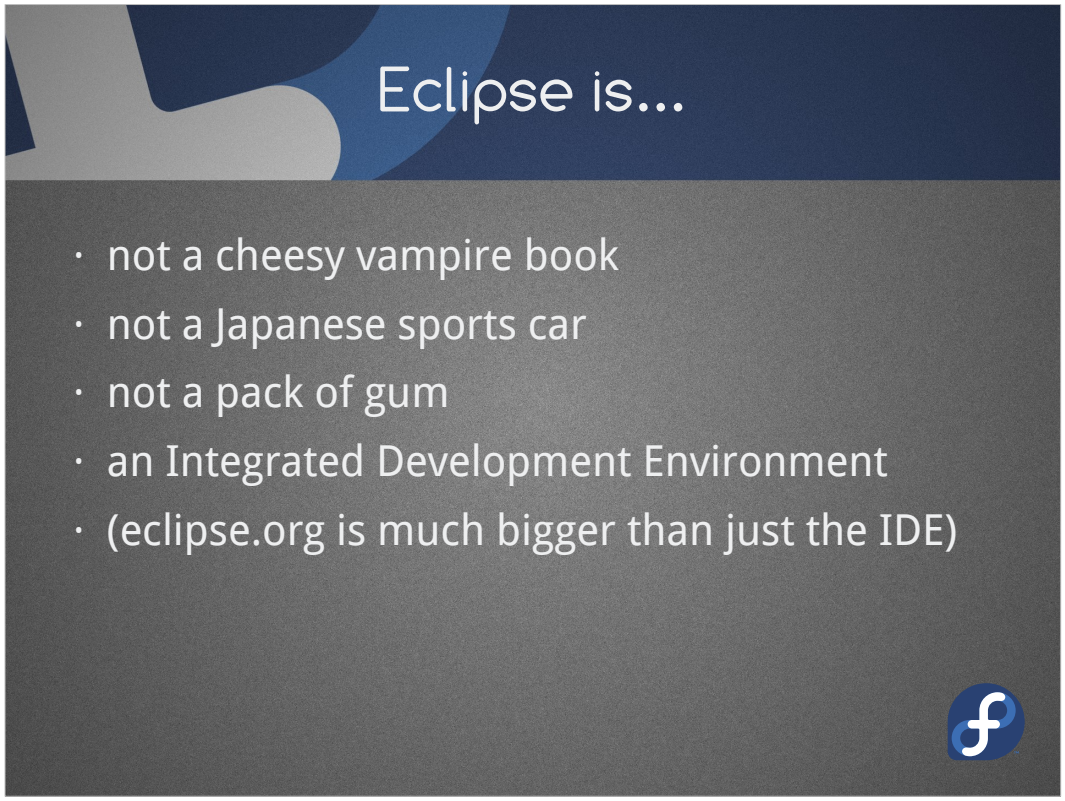
We have 2 hours, might be shorter.

Can have questions during or at the end.

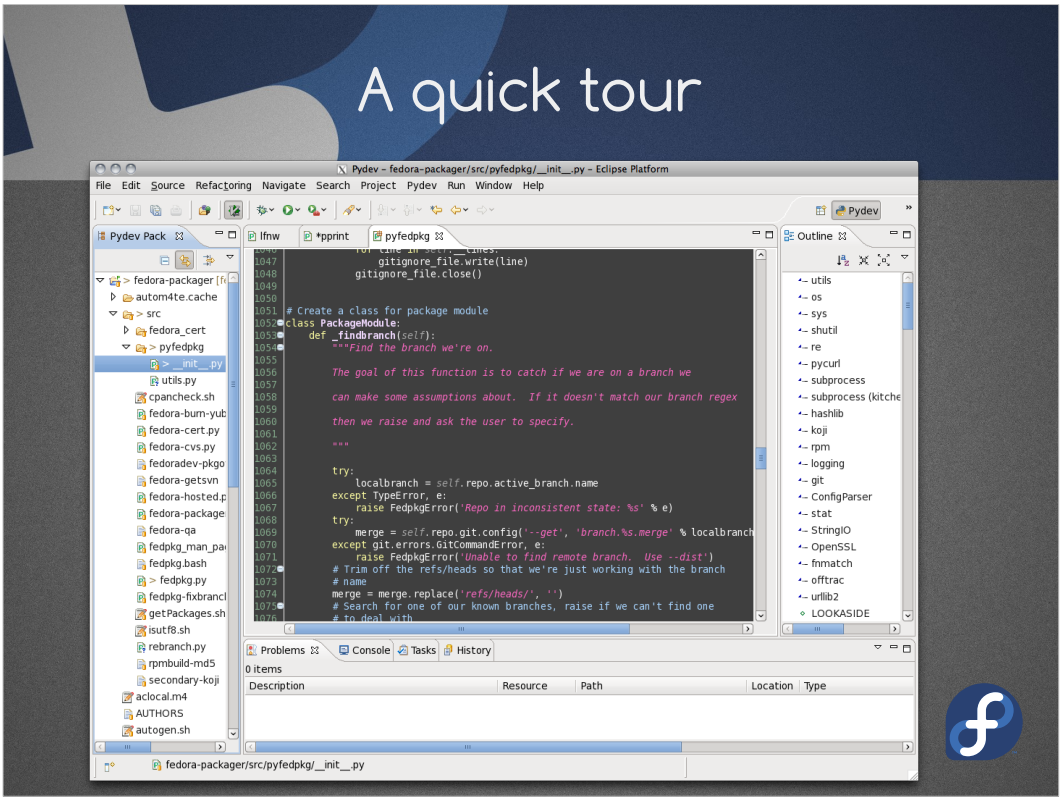
This does assume some working knowledge of python, git and vim. Knowledge of Eclipse is optional. Depending on time and pace I can dive further into topics to keep people from getting lost.



Does anybody not know what Eclipse is?



Kitchen Sink approach.



A few panes to look at

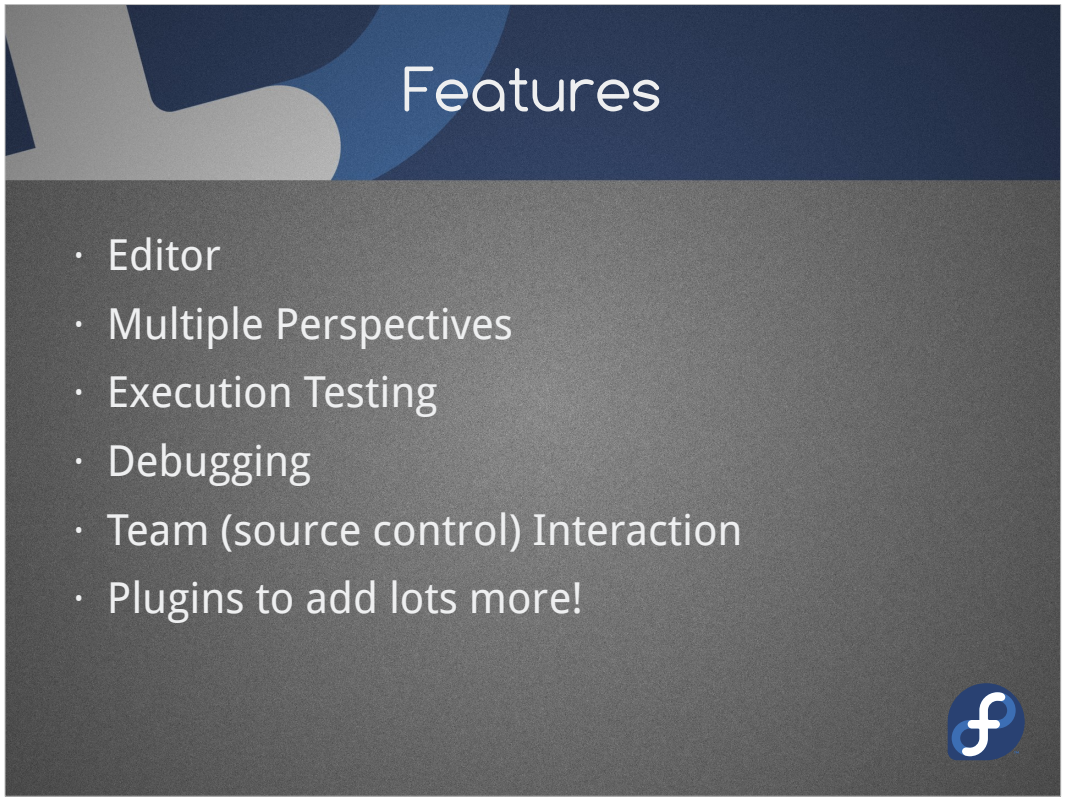
Project / file browser on left

Editor in the middle

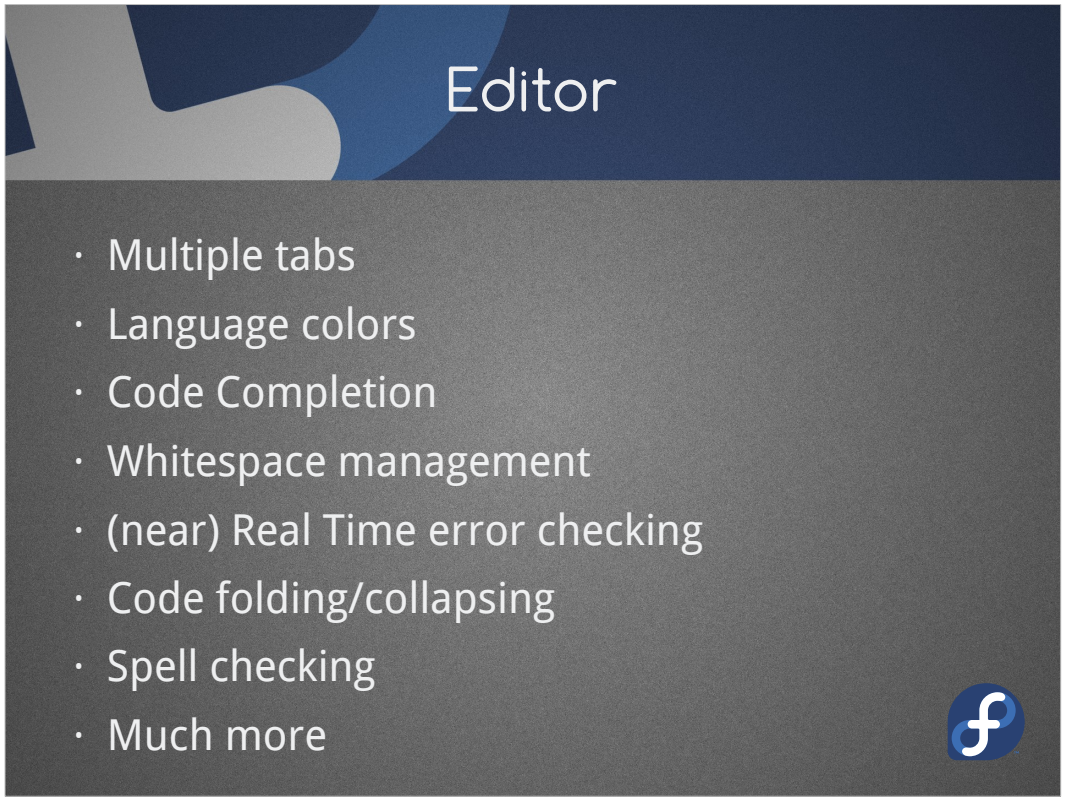
Outline on right

Various utilities on the bottom

Multiple perspectives

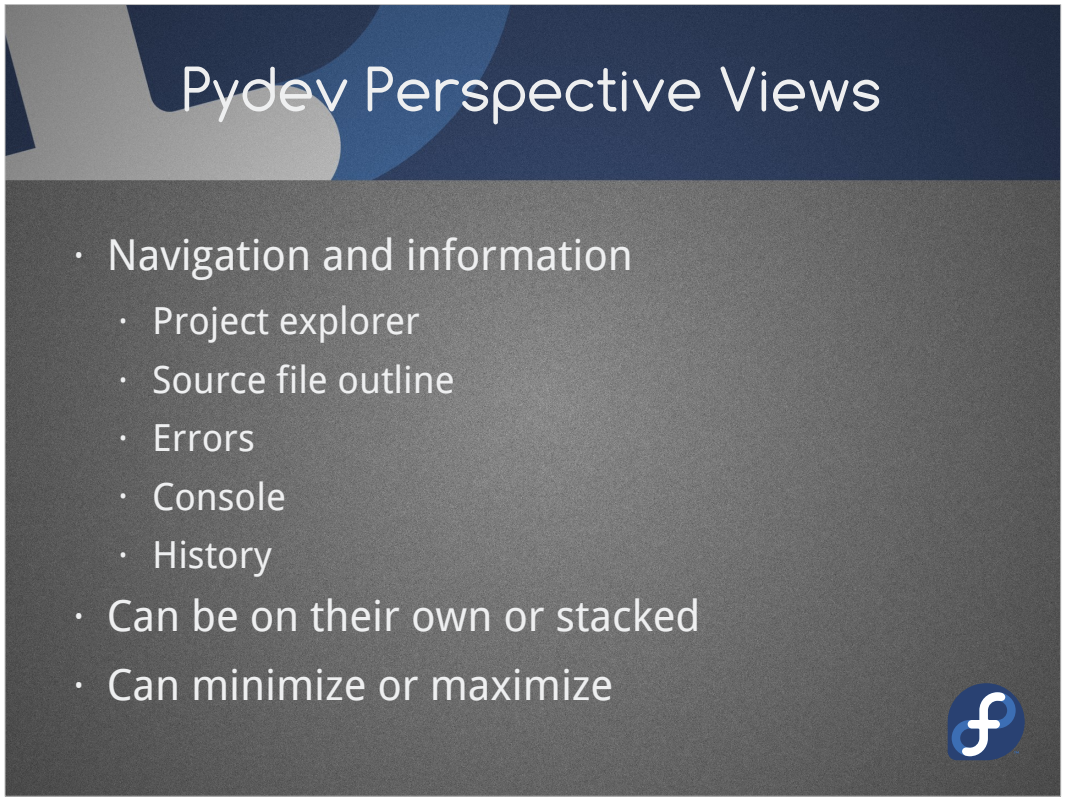


Perspectives define what is visible in the workbench, presets for editing, debugging, etc..



Editor is the main interface where you'll do most of the typing

List the editor main features



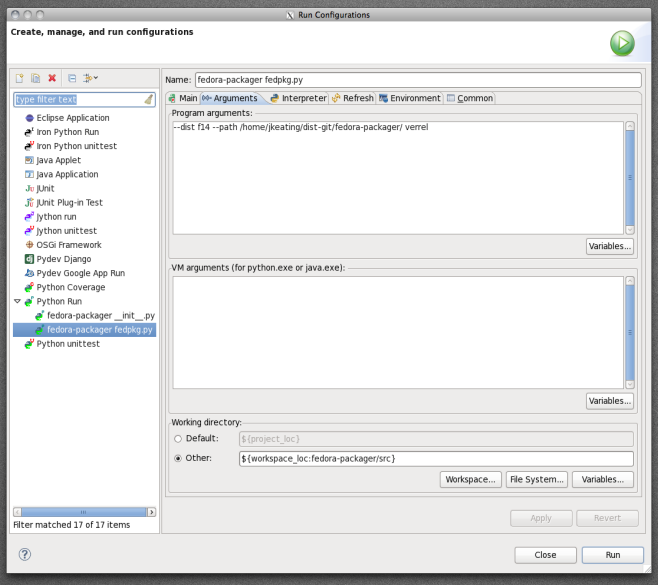
Views surround the editor and offer navigation and information

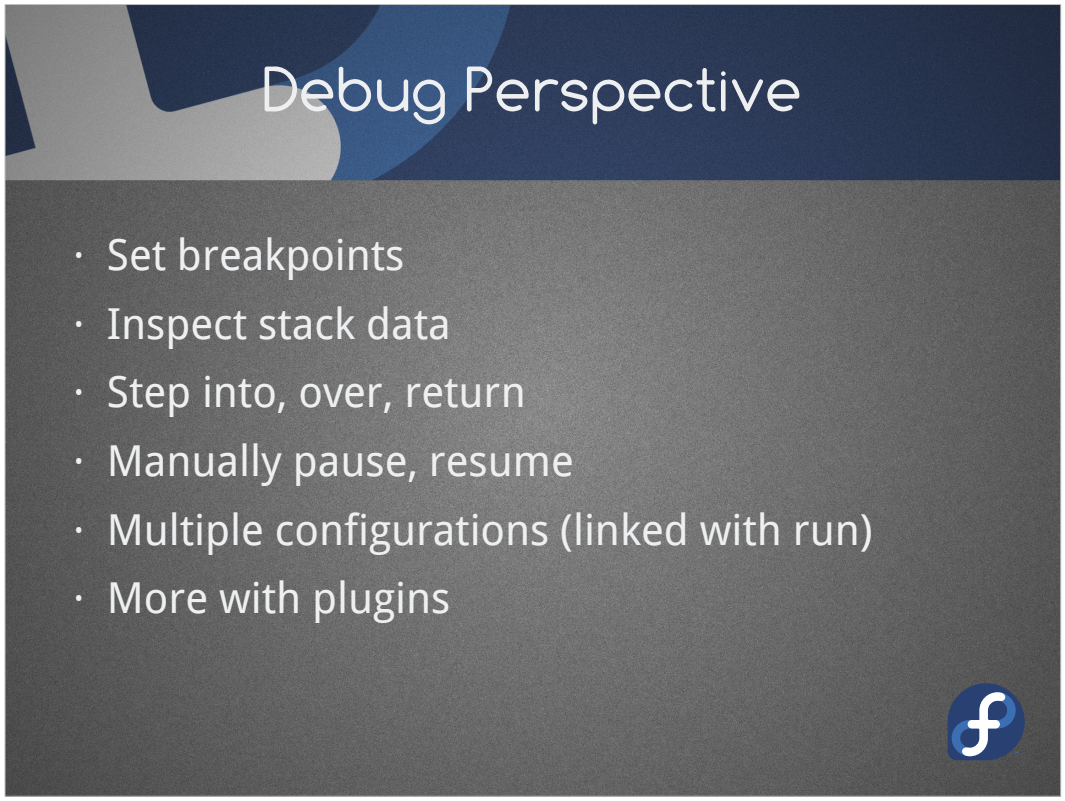
Perspectives are highly customizable

Execution Testing

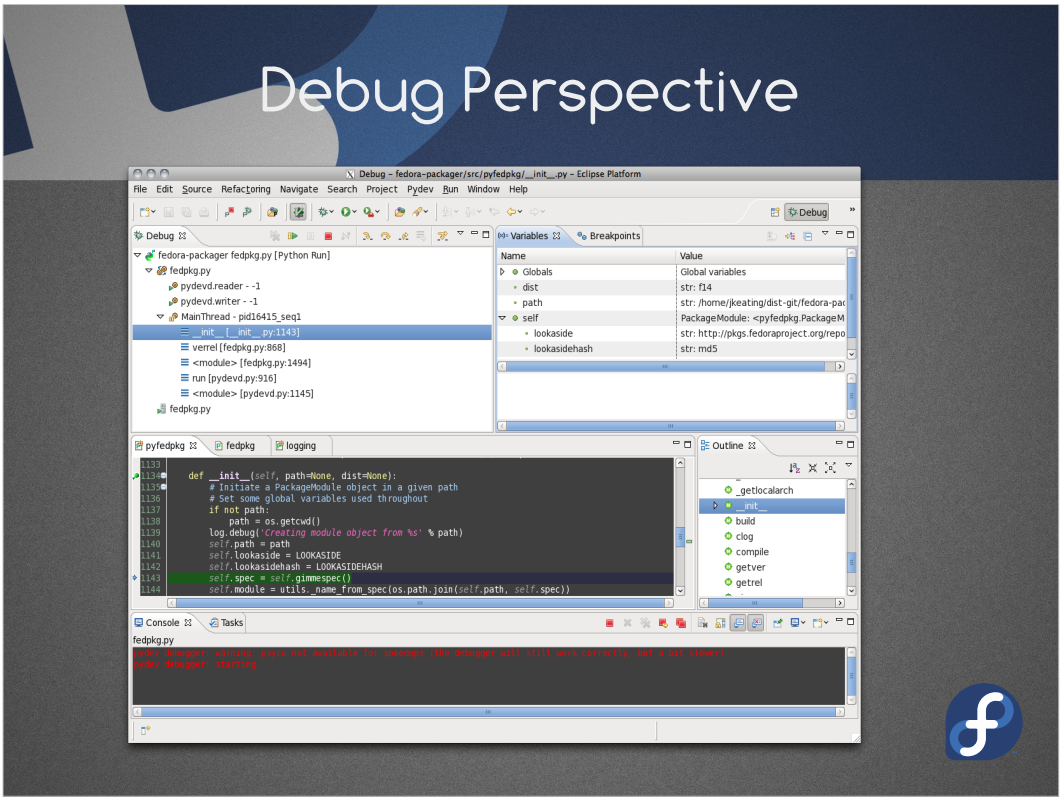
- Multiple configurations
- Custom app / interpreter arguments
- Console output
- Support for code coverage
- Support for Google App Run
- More with plugins

Execution Testing





Different views more tailored for debugging

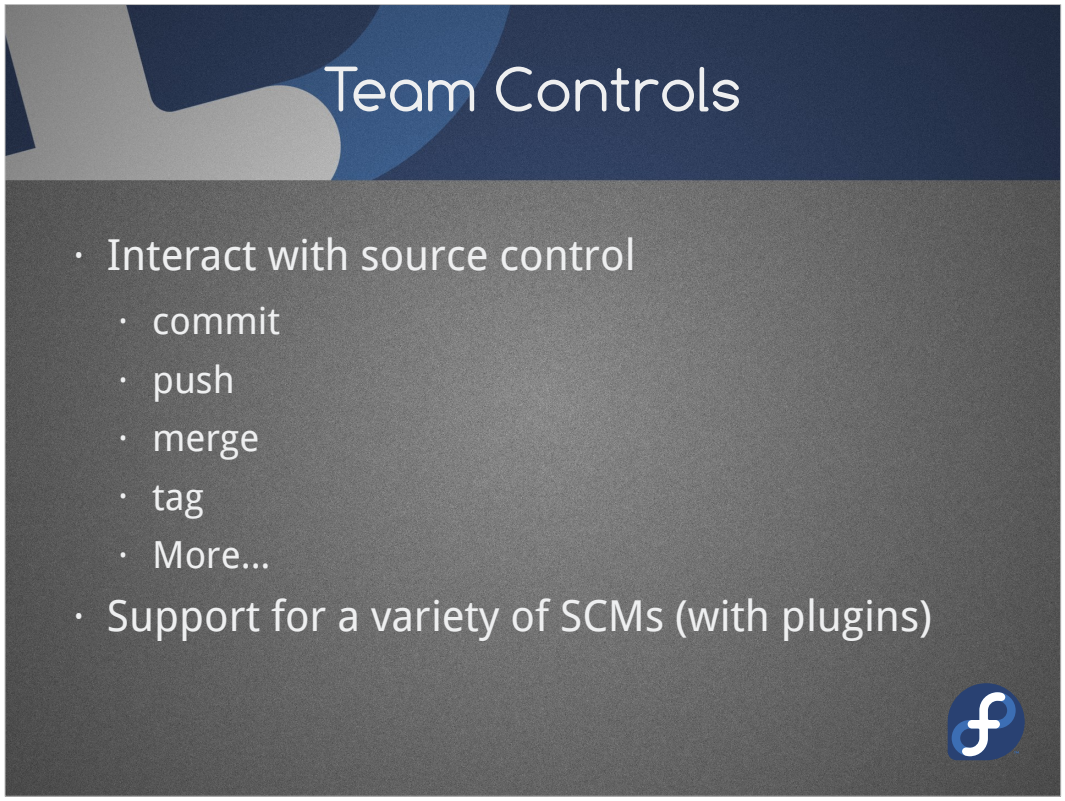


Thread data and flow manipulation

Variable data

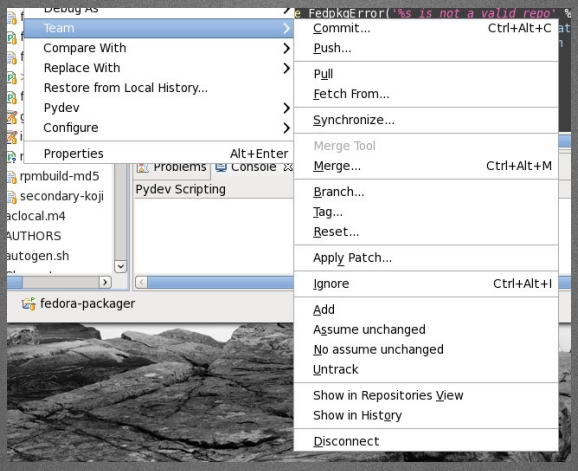
Smaller source window and overview, now with highlights to show current execution point

Console



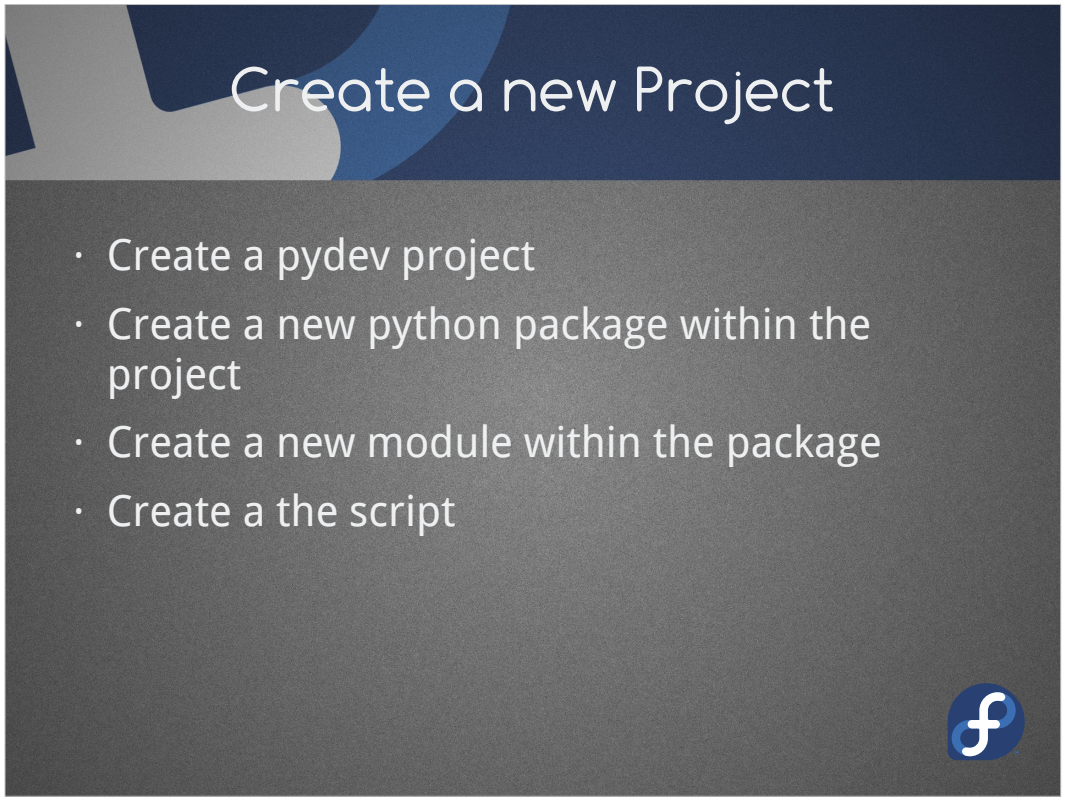
Team context menu will change depending on what SCM (if any) is in use

Team Controls



Lets move on to using Eclipse for writing some python code

Python support comes from the pydev project, packaged as eclipse-pydev



Pydev is the plugin to use for new python projects

When creating new packages, dot notation can be used to create submodules

When creating new modules, right clicking can help where the module winds up

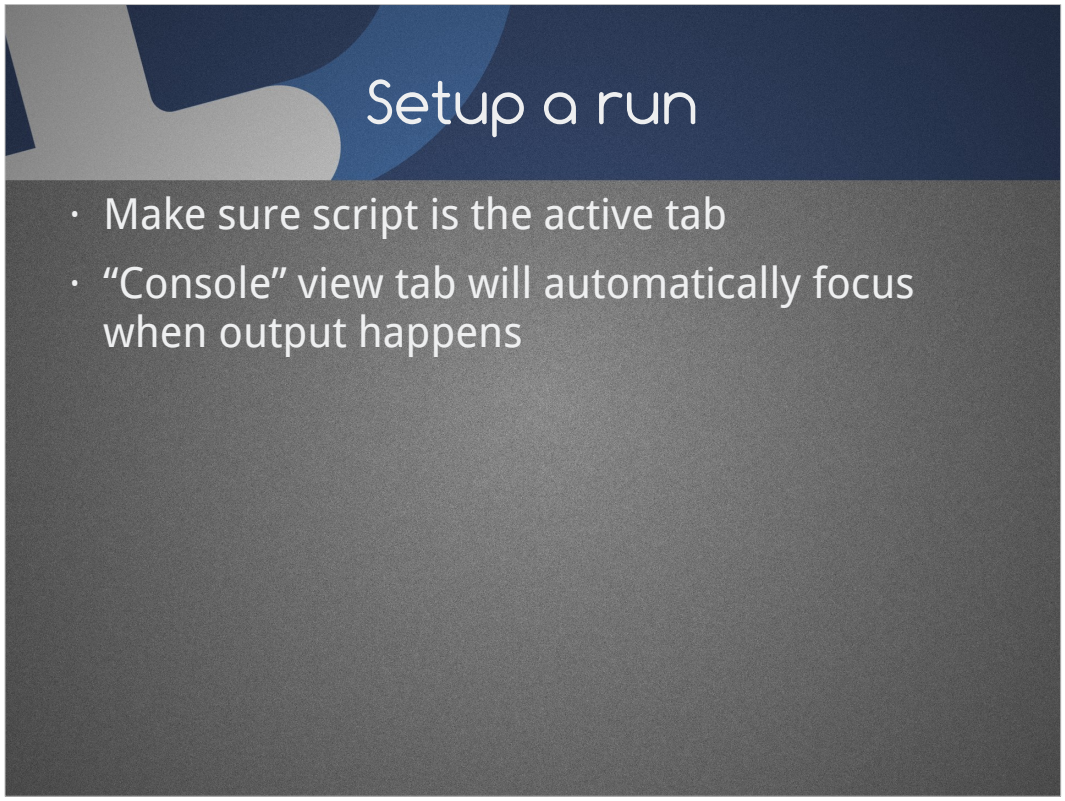
When creating new modules, templates can be used

Create the lfnw project

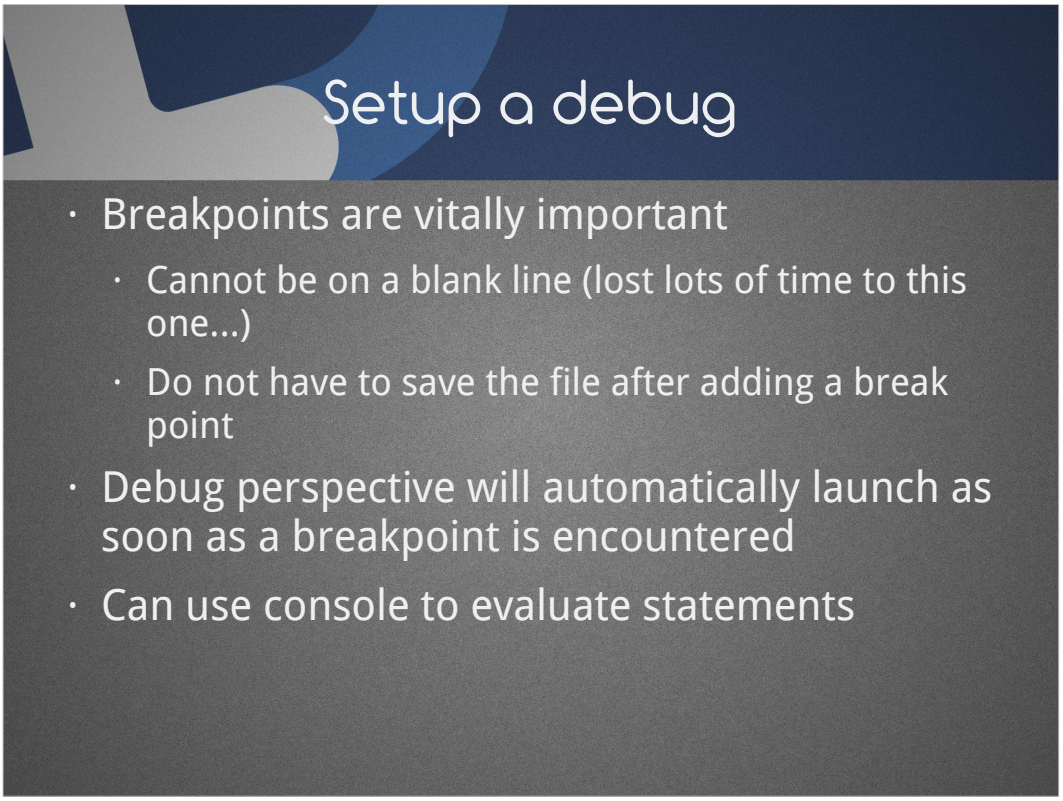
Create a package output.console
Create a module within console named pprint

Edit pprint to create a Print() class and a doprint() function within that prints a message

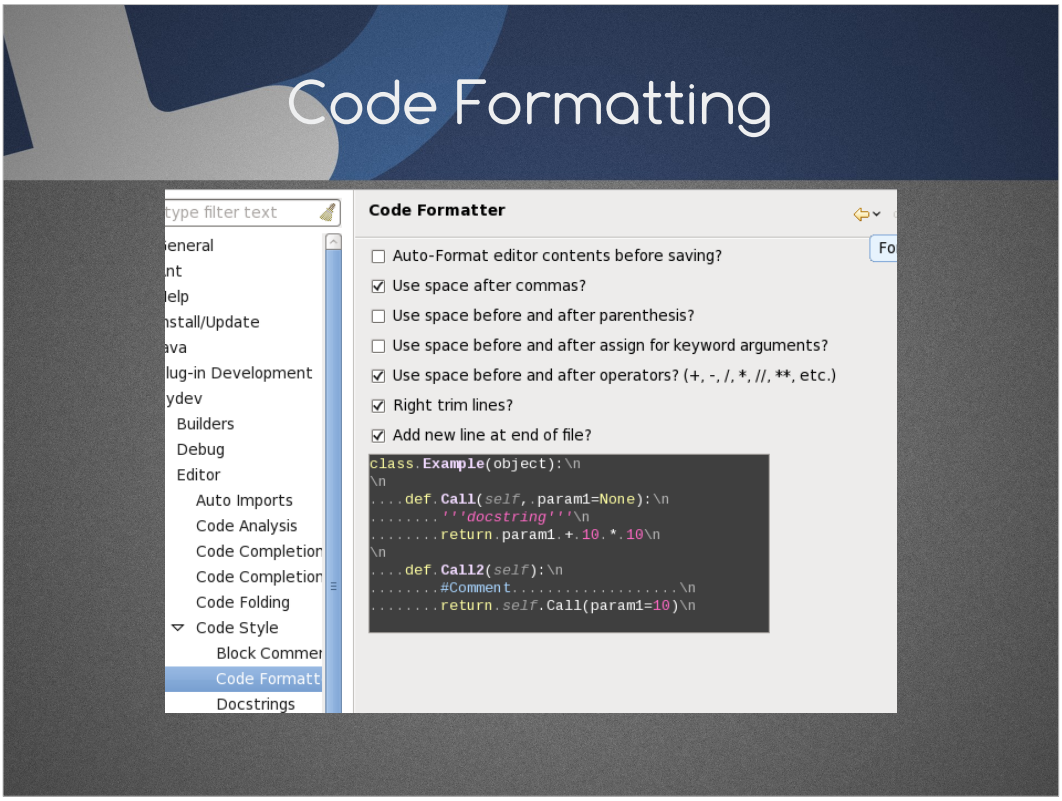
Create a module at top level using main template
Discuss how templates can be used and customized



Running this is easy, there are no options. Could define arguments to pass.



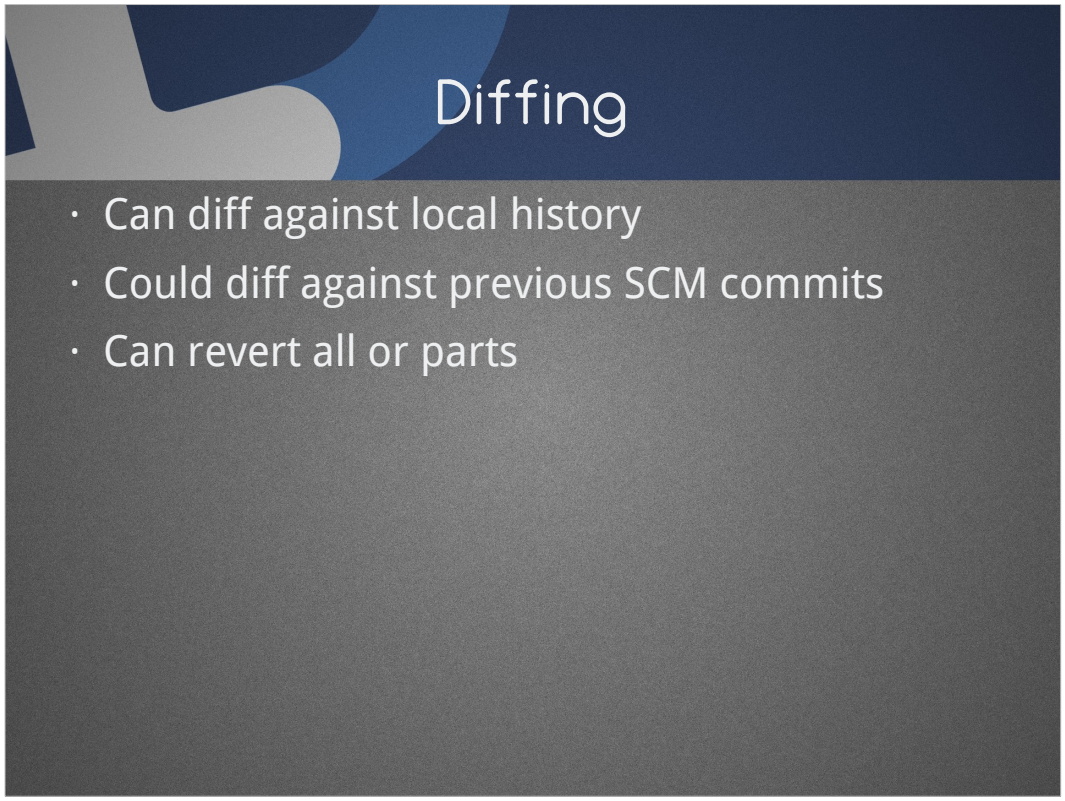
Insert a break point somewhere in the Print class



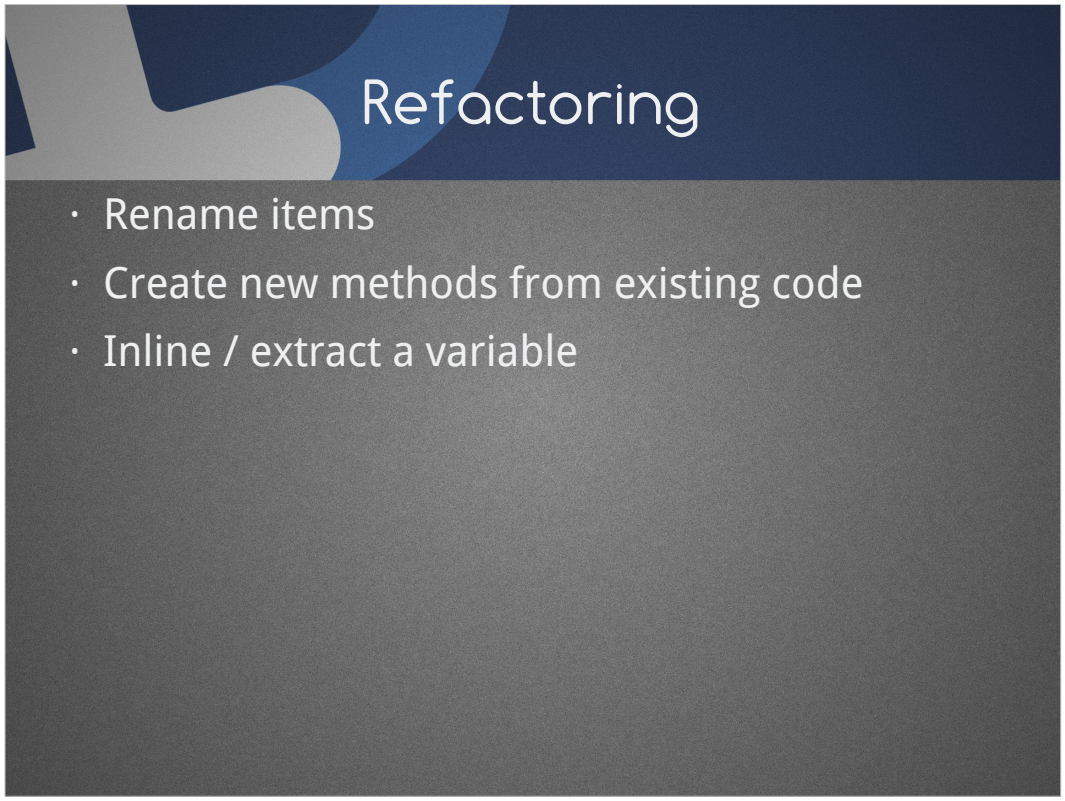
You can define what code formatting rules you'd like applied.

You can have it autoformat before saving, or do it manually.

Show running the code formatter on a file that has too many spaces, then diff.



Show diffing in the UI

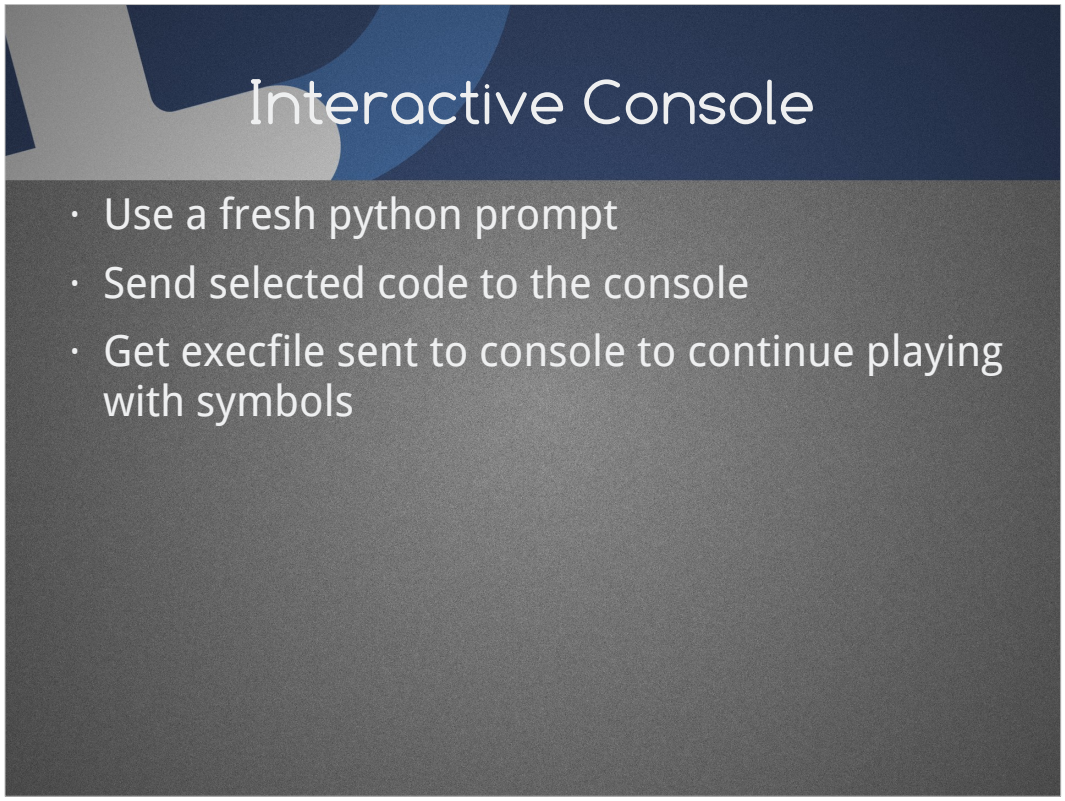


Rename an item and it will update all the references across the project

Highlight a set of code and turn it into a new method

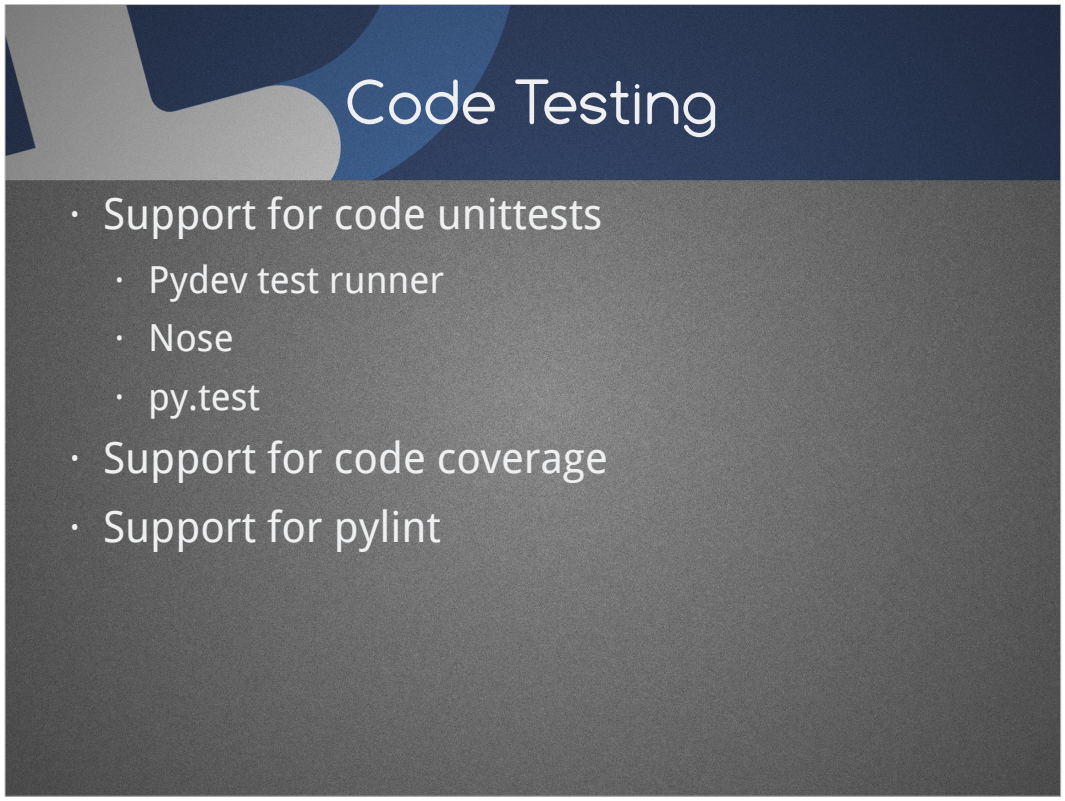
Collapse verbose code into more streamlined sets or vice versa

Show some examples



An interactive console can be used to play around with python, with some selected code sent to the console, or with an entire file sent and executed to allow you to play with the symbols and experiment.

Contents of the console can later be saved to a new file.



Eclipse can run your unittests for you using your choice of a few test runners.

This can be combined with code coverage information using the 'coverage' module. (show coverage demo)

Eclipse can also pylint your files as you edit them.

Add a test subpackage to output and create a subclass of unittest.TestCase (letting autoimport do its thing). Create a setUp class to create the module. Start adding test_foo for each function, running as coverage, checking the coverage each time. Don't forget if __name__ == '__main__'

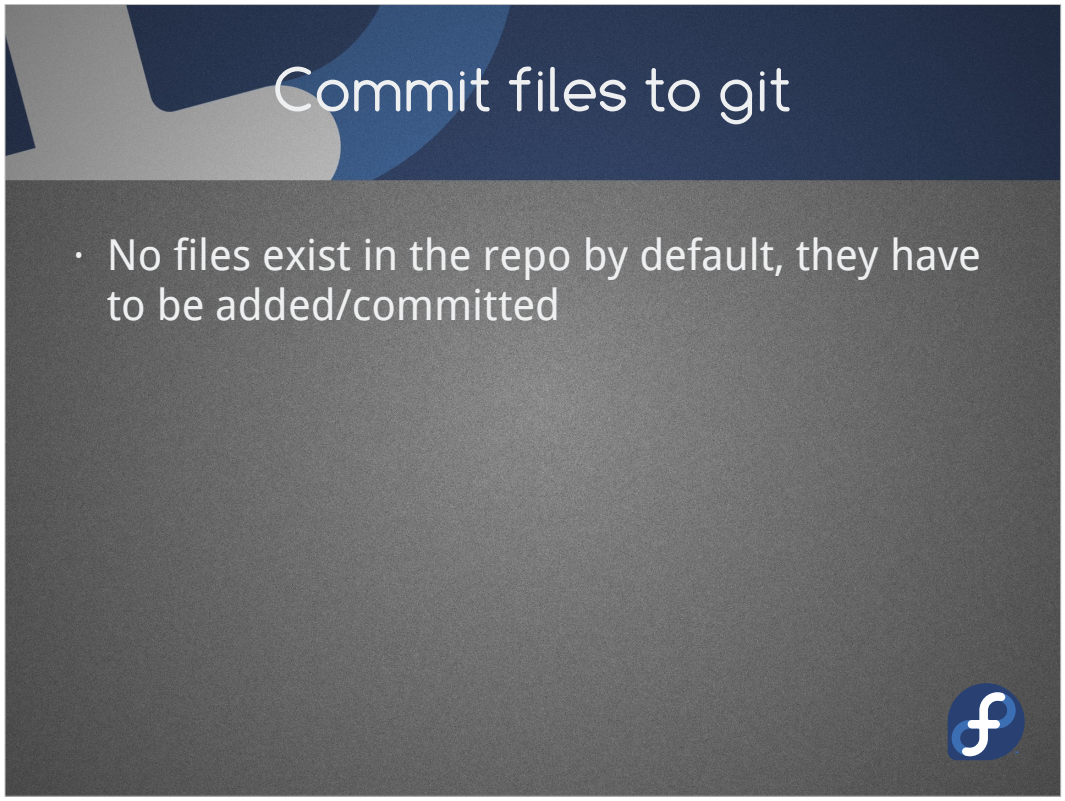
Now that we have some code in a project, lets start playing with source control to keep track of the changes we'll make.

Git interaction comes from the 'egit' plugin, which is packaged as 'eclipse-egit' in Fedora.

Show diffing in the UI

Use the team menu to share the project, which will allow you to create a new git repository of the project.

Now you can use the team menu to interact with git, and the project browser will have subtle graphical hints as to repository status



Show commit before add, then add then show commit again.

Use team menu to create and check out a new branch

In this branch add a new class method, show diffing before saving.

Commit the change and again show how you can look at the diff while in the commit screen

Create a patch from the commit in history view. Show difference between git exported and not. Still more useful to use git format-patch et al from the CLI

Add a change and then throw it away with reset, or with history viewing.

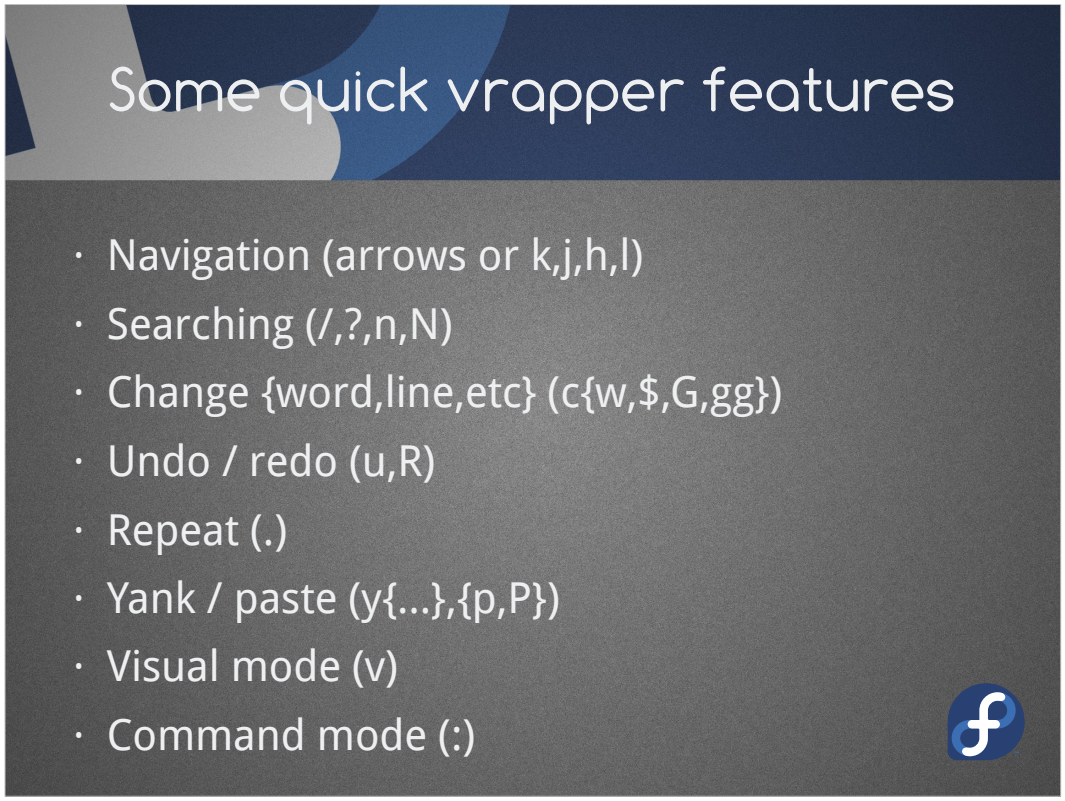
Checkout and add a change on master, then merge/rebase on branch (repos view), then merge on master

What good is a graphical editor if you're constantly fighting the keybindings?

Vim is awesome, would love to use it everywhere. Vim keybindings can be added to eclipse in a few different ways. The “best” “free” way I've found is with vrapper, packaged as eclipse-vrapper in Fedora.

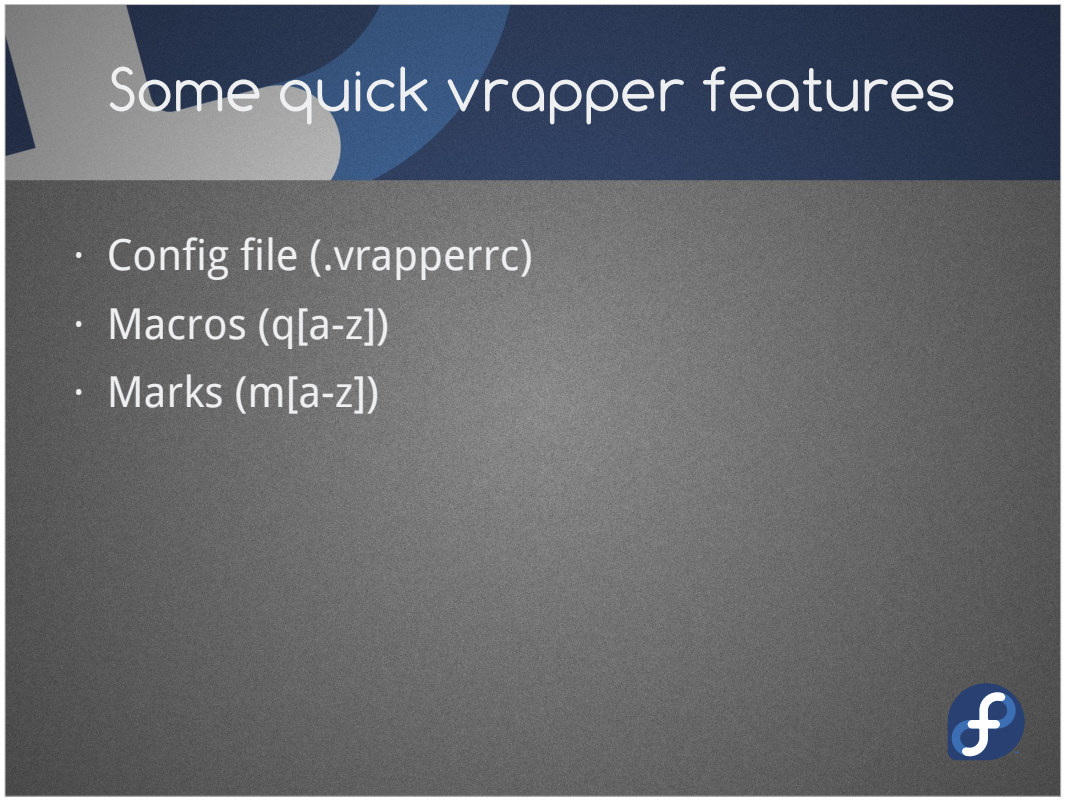
vrapper

- Wraps the current editor with vim like keybindings, rather than embedding vim itself
- Easy to turn on/off without restarting eclipse
- Still has command/insert modes
- Not all commands or key sequences work though, and a few bugs.



Move around, search around, change stuff, undo/redo the change, repeat an action, yank and paste (across tabs), show visual for yanking

Show some commands



There is a config file, it supports macros and marks for jumping

What's missing?

- Search and replace
- Regex searching
- Vim plugins

Summary

- Eclipse is a useful IDE
- Developing python in Eclipse is awesome
- Using git within Eclipse is handy
- Using vim within Eclipse is a godsend!

Questions?

CONTACT:
jkeating@redhat.com