Tools Introduction

Carl Friedrich Bolz, David Schneider
Dynamische Programmiersprachen
Heinrich-Heine-Universität Düsseldorf
Sommersemester 2010

Tools Used in the Exercises

- py.test testing framework `httg://pylib.org`
- Mercurial version control system
  `http://mercurial.selenic.com`

Version Control Using Mercurial

- Mercurial is a modern distributed Version Control System
- written in Python
- fairly fast
- many of the commands similar to SVN

Basic Commands

- `hg help <command>` gives help for a specific command
- `hg status` shows which files have changes in the working copy
- `hg diff` shows the modifications of the changed files
- `hg commit` commits the changes to the (local) repository
- `hg log` shows a list of past revisions
- `hg add <file>` adds a file to the repo on the next commit

Local vs. Remote Repositories

- in SVN, every commit changes the repository on the server
- in Mercurial every working copy has a full repository
- a commit only changes the local repository
- thus every working copy is also a branch
- to share changes with other people, more commands are needed

Commands not in SVN

- `hg push` pushes the changes in the local repo to the remote repo
- `hg pull` gets the changes in the remote repository that are not yet in the local repo

Conflicts

- push and pull only work when local and remote repos are "compatible enough"
- if somebody else pushed before you, you need to integrate his changes first
- `hg merge` does that. It always should be followed by a `hg commit`

Testing with py.test

- `py.test` is a testing tool to write automated tests for Python
- makes it quite easy to write and run tests
- automatic collection and running of tests

Writing of Tests

- every test is put into a top-level function without arguments
- the name of the function needs to start with `test_`
- to check things in the test, use the `assert <expr>` statement
- to check that an expression raises an exception, use `py.test.raises(<Exception>, "<expression">`

Running of Tests

- tests can be run with the command `py.test <filename(s)>`
- all the test functions in the file will be executed
- execution order is as function order in the file
- reports all the passing tests with a `"."`, all the failing ones with a `"F"`
- will get a traceback for failing tests

Disabling Tests

If a test is not working temporarily or under specific circumstances, it can be *skipped*.

```
def test_linux_specific():
    if sys.platform != "linux2":
        py.test.skip("only works under Linux")
    ... actual test
```

Expected Failures

A test that is failing for known reasons should not be committed to the repo. If that is necessary, it can be annotated to be an expected failure.

```
@py.test.mark.xfail
def test_not_implemented_yet():
    assert f(1) == 17
    assert f(2) == 82
```

Commandline Options

Some useful options that the `py.test` utility takes as arguments:

- `-v` verbose, prints the names of all run functions
- `-x` stop test run at first failure
- `--pdb` on every failure, drop into a debugger instance (`pdb` = python debugger)
- `-k <keyword>` run only tests that match the keyword