Pretor User's Manual

For Instructors

Contents

Chapter 1

Introduction

1.1 What is Pretor?

Pretor is an automated "grading assistant". It is a program which can help you manage your student's submissions, your grades and feedback, and enable you to easily create automation. There are several ways to use Pretor...

i. As a tool for facilitating manual grading. In it's default state, Pretor will manage student submissions, allow you to interact them in a Bash shell, record your scores in a simple TOML format, archive your grades and feedback for posterity, and export a spreadsheet you can upload into your university's LMS.

ii. As a platform for machine-assisted grading. It's easy to write your own plugins or other tools; you can then use Pretor as a tool to interactively orchestrate your automation.

iii. As a library for fully-automated grading. Pretor provides powerful primitives that could be used as the basis for an unattended grading system. The interactive grading REPL also supports the execution of script files, allowing it to be run in a headless unattended mode.

There are three major components of Pretor:

`pretor-psf` is used by students to generate PSFs (Pretor Submission Files), which they can submit through whatever mechanism you find appropriate.

`pretor-grade` implements an interactive REPL that enables a grader to efficiently iterate through many PSFs in sequence. `pretor-grade` ultimately produces more PSFs as output, which have the grades and other feedback the grader assigns "burned in" to them.

`pretor-export` is a tool which operates on the PSFs produced by `pretor-grade` and generates output files that can be read by humans, or imported by an LMS for bulk grading.

1.2 Understanding the Pretor Data Model

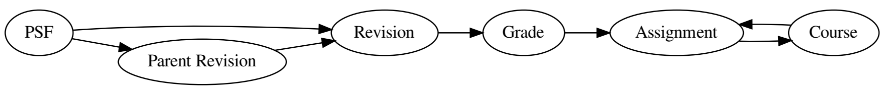

Figure 1.1: High-level overview of the Pretor data model

Understanding Pretor's data model is critical to make efficient use of its features. Fortunately, Pretor has a relatively simple data model that aligns closely with how courses, grades, and submissions are intuitively reasoned about.

- A PSF contains one or more revisions.
- A revision may contain zero or one grades.
- A grade is associated with an assignment.
- An assignment is associated with a course.
- A course is associated with one or more assignments.

Aside: for technical reasons, all PSF files contain serialized copies of the assignment and course information for each grade they contain. This is because a grade is meaningless without a rubric (to weight each category score), and a course (to determine the weight of the assignment overall).

You as the instructor interact with the data model in several ways. One important way is by writing a **course definition file**, which defines the set of assignments in your course and their relative weights, as well as the rubric categories for each assignment. This is used by `pretor-grade` to compute the scores for each assignment you grade, and by `pretor-export` to generate appropriate score values.

You will also interact with the data model by grading assignments. Each time you grade an assignment, you are creating a **revision** in the PSF the student turned in, creating a **grade**, and attaching the grade to the revision.

Key Concept: Pretor has it's own internal revision control system. Student-generated PSFs contain an initial "submission" revision. When you grade a PSF, you create a "grade" revision, which can include both changes to score or other metadata, as well as changes to the student's submitted code. This is valuable because it makes it easy to track changes made to get student code to compile, and allows you to make in-line comments within the student's code. You can even revise an existing grade revision later if you realize you made a mistake, which would create a third revision. Arbitrarily many grade revisions may be made.

1.3 Pretor Workflow

Using Pretor is straightforward, barring additions made by third-party plugins, a typical Pretor grading workflow looks like this:

1. Download PSFs for a specific assignment from your institution's LMS
2. Run `pretor-grade` on the downloaded files, assigning a grade to each, this produces more PSFs which contain both the student's original submission and your modifications and feedback
3. Run `pretor-export` on the PSFs generated in the previous steps to generate a CSV file appropriate for upload into your LMS

Chapter 2

Grading With Pretor

2.1 Grading Basics

You can begin an interactive grading session with the command `pretor-grade`. `pretor-grade` has many useful parameters you should explore[1], but the most important are `--ingest`, `--outputdir`, and `--coursepath`.

`--ingest` is used to specify a directory where you have downloaded your PSFs. This directory is searched recursively for `*.psf`, all of which are loaded into your grading REPL before it begins. You can also ingest PSFs after launching via the `ingest` command.

`--outputdir` when you finish grading an assignment and mark it as finalized, the resulting PSF will be stored in this directory. If unspecified, they'll be placed in your working directory.

`--coursedir` specifies the directory where your course definition file(s) are stored. When you begin grading a PSF, the course name and assignment name specified in the submission's `pretor.toml` are looked up by recursive search through every TOML in your configured coursedir. When a matching file is found, it is loaded and used to pre-populate the `grade.toml` that you will use to enter your scores. If you don't specify this, your working directory will be used.

You should be greeted by a prompt that looks like this:

```
PRETOR version 0.0.1 interactive grading shell.
grader>
```

You can enter the `help` command here to see a list of all commands available in the REPL, and `help <command name>` to see documentation for a specific command.

While there are many useful commands available, the most essential are:

`loaded` displays a list of PSF files that have been loaded

`current` show information about the PSF you are working on right now

`next` load the next un-graded PSF that is loaded

`interact` drop to a Bash shell to grade the PSF

`showgrade` show the score card for the current PSF

[1]see `pretor-grade --help`

`finalize` save your changes to the PSF and write it out into the configured output directory

With only these commands, you can perform all grading tasks with Pretor.

Let's look at an example grading session with Pretor:

```
$ pretor-grade --ingest submissions
INFO: Loading PSF file 'submissions/Spring 1973-ABC123-2-jsmith-Assignment 1.psf'
INFO: Loading PSF file 'submissions/Spring 1973-ABC123-2-jdeer-Assignment 1.psf'
INFO: Loading PSF file 'submissions/Spring 1973-ABC123-2-jdoe-Assignment 1.psf'
PRETOR version 0.0.1 interactive grading shell.
grader> loaded
       0: Spring 1973-ABC123-2-jsmith-Assignment 1.psf
       1: Spring 1973-ABC123-2-jdeer-Assignment 1.psf
       2: Spring 1973-ABC123-2-jdoe-Assignment 1.psf
       grader> next
<PSF ID=UUID('7c6f7597-aba0-43bd-bee6-a829943bfcd7')>
semester      Spring 1973
section       2
assignment    Assignment 1
group         jsmith
course        ABC123
timestamp     2019-02-06 19:30:33.046311
archive_name  submissions/Spring 1973-ABC123-2-jsmith-Assignment 1.psf
PSF has NOT been graded
grader> interact
INFO: dropping you to a shell: bash --norc
grading Assignment 1 by jsmith $ tree
.
├── grade.toml
└── submission
    ├── doc
    │   └── HOWTO.txt
    ├── hello.c
    ├── Makefile
    ├── pretor.toml
    ├── util.c
    └── util.h

2 directories, 7 files
grading Assignment 1 by jsmith $ cat grade.toml
feedback = ""
bonus_multiplier = 0.0
bonus_marks = 0
bonus_score = 0.0
penalty_multiplier = 0.0
penalty_marks = 0
penalty_score = 0.0
assignment_name = "Assignment 1"

[categories]
correctness = 70
style = 30
```

The `grade.toml` file is perhaps the most important thing to notice here. This is how you input the grade you would like to assign. When you interact with a PSF for the first time, this file is populated with the maximum values for each category as determined by your course definition file. In other words, every submission starts out with a 100% score, and modifying the values in the [categories] section allows you to change the submission's score.

```
grading Assignment 1 by jsmith $ exit
exit
INFO: shell session terminated
grader> showgrade
SCORECARD FOR ABC123: Assignment 1

CATEGORY     MARKS  MAX MARKS  PERCENT SCORE
correctness  70     70         100.00%
style        30     30         100.00%

OVERALL MARKS: 100
MAXIMUM OVERALL MARKS: 100
RAW SCORE: 100.00%

OVERALL SCORE: 100.00%

grader> finalize
INFO: writing to 'Spring 1973-ABC123-2-jsmith-Assignment 1.psf'
grader> exit
$ pretor-psf --scorecard --input Spring\ 1973-ABC123-2-jsmith-Assignment\ 1.psf
SCORECARD FOR ABC123: Assignment 1

CATEGORY     MARKS  MAX MARKS  PERCENT SCORE
correctness  70     70         100.00%
style        30     30         100.00%

OVERALL MARKS: 100
MAXIMUM OVERALL MARKS: 100
RAW SCORE: 100.00%

OVERALL SCORE: 100.00%
```

Notice that the assigned grade is saved out to disk as soon as finalize is issued, and can be retrieved later using pretor-psf.

Now consider an example where we have graded several PSFs already, and want to see the overall score on each. For this, we can use the pretor-export command:

```
$ ls
 coursedefs        'Spring 1973-ABC123-2-jdeer-Assignment 1.psf'    submissions
 README.md         'Spring 1973-ABC123-2-jdoe-Assignment 1.psf'
 sample_solutions  'Spring 1973-ABC123-2-jsmith-Assignment 1.psf'
 $ pretor-export --input '*.psf' --table
Spring 1973  ABC123  2  jsmith  90
Spring 1973  ABC123  2  jdeer   50
Spring 1973  ABC123  2  jdoe    90  submitted late, -10%
```

2.2 Bonuses, Penalties & Grade Calculation

Each grade contains a number of **categories**. A category has a maximum number of marks[2] and a number of assigned marks. A grade's raw score is simply $\frac{\sum \text{marks}_{\text{max}}}{\sum \text{marks}_{\text{assigned}}}$. The function of **categories** is to allow the instructor to provide greater granularity to assigned scores, and to accurate codify the categories of an assignment's rubric.

category	$\text{marks}_{\text{max}}$	$\text{marks}_{\text{assigned}}$
all test cases pass	50	40
correct style	30	25
code is documented	30	30

Figure 2.1: Example categories and assigned scores

Considering the example shown in figure 2.1, the maximum raw marks for this grade would be $50 + 30 + 30 = 110$, and the assigned marks would be $40 + 25 + 30 = 95$, for a **raw score** of $\frac{95}{110} \approx 0.863$. Note that the total number of maximum marks does not need to sum to 100 (this example was deliberately constructed to demonstrate this).

While bonuses may be given by simply entering marks higher than the maximum for a given category, this is not the suggested approach, as Pretor includes dedicated facilities for specifying bonuses (and penalties). The final score of a given grade is computed as: $g = \frac{m+b_m-p_m}{M} \cdot (1.0 + b - p) + B - P$ where:

- g is the final percent score in 0..1 (scores of higher than 1 may be possible with bonus)
- m is the number of earned marks on the assignment (sum of category scores)
- b_m is the number of bonus marks on the assignment
- p_m is the number of penalty marks on the assignment
- M is the maximum number of marks on the assignment (sum of category maxes)
- b is the bonus multiplier
- p is the penalty multiplier
- B is the score bonus
- P is the score penalty

The `grade.toml` file generated while `interact`-ing with a PSF with `pretor-grade` will automatically have appropriate fields for each type of bonus and penalty initialized to values that will provide no bonus and no penalty.

Finally, to handle cases where the provided facilities are insufficient for assigning the desired grade, a `grade.toml` file may also specify a `override` field, which if provided, is unconditionally used as the final grade. Note that the `override` field is on a scale of 0..1, such that a value of 1 would be a 100% score, although `override` is not bounded by 0..1 (i.e. scores above 100% or lower than 0% may be assigned. This convention is used throughout Pretor except where noted.

Hint: Multiple penalties and bonuses can be mix-and-matched. For the sake of clarity, the example shown here only shows one penalty and one bonus applied at a time.

An example grading session is shown below demonstrating assigning bonuses, penalties, and overrides:

[2]Maximum marks on a category is determined by the associated course and assignment

2.2.1 Example Grading Session

Initial Grade

```
$ pretor-grade --ingest submissions --coursepath coursedefs
./
INFO: Loading PSF file 'submissions/Spring 1973-ABC123-2-cad-Assignment 1.psf'
PRETOR version 0.0.3-dev interactive grading shell.
grader> next
<PSF ID=c1a0b0a3-1972-4974-a890-25f97c270b4c>
semester        Spring 1973
section         2
assignment      Assignment 1
group           cad
course          ABC123
timestamp       2019-02-10 14:02:18.709983
pretor_version  0.0.3-dev
archive_name    submissions/Spring 1973-ABC123-2-cad-Assignment 1.psf
PSF has NOT been graded
grader> interact
INFO: dropping you to a shell: bash --norc
grading Assignment 1 by cad $ vim grade.toml
grading Assignment 1 by cad $ cat grade.toml
feedback = ""
bonus_multiplier = 0.0
bonus_marks = 0
bonus_score = 0.0
penalty_multiplier = 0.0
penalty_marks = 0
penalty_score = 0.0
assignment_name = "Assignment 1"

[categories]
correctness = 50
style = 20
grading Assignment 1 by cad $ exit
exit
INFO: shell session terminated
grader> showgrade
SCORECARD FOR ABC123: Assignment 1

CATEGORY     MARKS  MAX MARKS  PERCENT SCORE
correctness  50     70         71.43%
style        20     30         66.67%

OVERALL MARKS: 70
MAXIMUM OVERALL MARKS: 100
RAW SCORE: 70.00%

OVERALL SCORE: 70.00%
```

Assigning a Bonus

```
grader> interact
INFO: dropping you to a shell: bash --norc
```

```
grading Assignment 1 by cad $ vim grade.toml
grading Assignment 1 by cad $ cat grade.toml
feedback = ""
bonus_multiplier = 0.0
bonus_marks = 10
bonus_score = 0.0
penalty_multiplier = 0.0
penalty_marks = 0
penalty_score = 0.0
assignment_name = "Assignment 1"

[categories]
correctness = 50
style = 20
grading Assignment 1 by cad $ exit
exit
INFO: shell session terminated
grader> showgrade
SCORECARD FOR ABC123: Assignment 1

CATEGORY     MARKS  MAX MARKS  PERCENT SCORE
correctness  50     70         71.43%
style        20     30         66.67%
BONUS MARKS  10     --         --

OVERALL MARKS: 70
MAXIMUM OVERALL MARKS: 100
RAW SCORE: 70.00%
RAW SCORE NET OF BONUS/PENALTY MARKS: 80.00%

OVERALL SCORE: 80.00%
```

Assigning a Penalty

```
grader> interact
INFO: dropping you to a shell: bash --norc
grading Assignment 1 by cad $ vim grade.toml
grading Assignment 1 by cad $ cat grade.toml
feedback = ""
bonus_multiplier = 0.0
bonus_marks = 10
bonus_score = 0.0
penalty_multiplier = 0.1
penalty_marks = 0
penalty_score = 0.0
assignment_name = "Assignment 1"

[categories]
correctness = 50
style = 20
grading Assignment 1 by cad $ exit
exit
INFO: shell session terminated
grader> showgrade
```

```
SCORECARD FOR ABC123: Assignment 1

CATEGORY     MARKS  MAX MARKS  PERCENT SCORE
correctness  50     70         71.43%
style        20     30         66.67%
BONUS MARKS  10     --         --

OVERALL MARKS: 70
MAXIMUM OVERALL MARKS: 100
RAW SCORE: 70.00%
RAW SCORE NET OF BONUS/PENALTY MARKS: 80.00%

PENALTY MULTIPLIER: 0.10
SCORE NET OF BONUS/PENALTY MULTIPLIER: 72.00%

OVERALL SCORE: 72.00%
```

Assigning an Override

```
grader> interact
INFO: dropping you to a shell: bash --norc
grading Assignment 1 by cad $ vim grade.toml
grading Assignment 1 by cad $ cat grade.toml
feedback = ""
bonus_multiplier = 0.0
bonus_marks = 10
bonus_score = 0.0
penalty_multiplier = 0.10
penalty_marks = 0
penalty_score = 0.0
assignment_name = "Assignment 1"
override = 0.9

[categories]
correctness = 50
style = 20
grading Assignment 1 by cad $ exit
exit
INFO: shell session terminated
grader> showgrade
SCORECARD FOR ABC123: Assignment 1

CATEGORY     MARKS  MAX MARKS  PERCENT SCORE
correctness  50     70         71.43%
style        20     30         66.67%
BONUS MARKS  10     --         --

OVERALL MARKS: 70
MAXIMUM OVERALL MARKS: 100
RAW SCORE: 70.00%
RAW SCORE NET OF BONUS/PENALTY MARKS: 80.00%

PENALTY MULTIPLIER: 0.10
SCORE NET OF BONUS/PENALTY MULTIPLIER: 72.00%
SCORE HAS BEEN OVERRIDDEN BY GRADER

OVERALL SCORE: 90.00%
```

2.3 Constructing `pretor.toml` Files

2.4 Constructing Course Definitions

As the instructor for a course, you will need to write a course definition file for your course. This codifies the set of assignments (or other graded materials), their relative weights within the course, and their rubrics.

For a Pretor grade to be meaningful, it must be associated with a **course definition**. When you `interact` with a PSF in `pretor-grade`, the course name field (`course`) of it's `pretor.toml` file is searched for among all course definition files in the specified course directory for one with a matching `name` field.

Note: When you grade a PSF with `pretor-grade`, the course definition used at the time is "burned in" to the output PSF. This is to ensure that future modifications to the course definition will not retro-actively change existing grades.

A course definition file may have any name the author finds descriptive, but must have the extension `.toml`. At a minimum a course definition must define a section named `course` containing a key named `name`, which must specify the course name (which is matched against the `course` in `pretor.toml` files). The `course` section may also optionally contain a `description` field, which is an arbitrary human-readable string for reference purposes.

All other sections in the course definition file may have arbitrary names, and correspond to assignment rubrics. Each such section must define a `name` field which is matched against the `assignment` field in `pretor.toml` files, as well as a floating point `weight` field defining the assignment's weight[3]. An optional `description` field may be defined as well. Finally, each additional field specified is assumed to define the maximum number of marks on a rubric category.

Best Practice: is to store all of your course definition files in a single directory, so you can easily reference them via `pretor-grade`.

An example course definition file is shown below:

```
[course]
name = "ABC123"
description = "Imaginary course for testing Pretor."

[assignment_1]
name = "Assignment 1"
description = "Description for the first assignment"
weight = 0.05
correctness = 70
style = 30

[assignment_2]
name = "Assignment 2"
description = "Description for the second assignment"
```

[3]Course definition `weight` fields assume that a perfect score in the course is 1.0.

```
weight = 0.1
correctness = 70
style = 30

[assignment_3]
name = "Assignment 3"
description = "Description for the third assignment"
weight = 0.1
correctness = 70
style = 30

[midterm]
name = "Midterm"
weight = 0.2
problem_1 = 10
problem_2 = 10
problem_3 = 10
problem_4 = 70

[quiz1]
name = "Quiz 1"
weight = 0.1
problem_1 = 10
problem_2 = 10
problem_3 = 10

[quiz2]
name = "Quiz 2"
weight = 0.1
problem_1 = 10
problem_2 = 10
problem_3 = 10
problem_4 = 10
problem_5 = 10

[final]
name = "Final Exam"
weight = 0.35
problem_1 = 10
problem_2 = 10
problem_3 = 10
problem_4 = 30
problem_5 = 20
problem_6 = 20
problem_7 = 10
```