# Pretor User's Manual

For Instructors

# Contents


**1 Introduction** **5**
- 1.1 What is Pretor? . . . 5
- 1.2 Understanding the Pretor Data Model . . . 5
- 1.3 Pretor Workflow . . . 6

**2 Grading With Pretor** **7**
- 2.1 Grading Basics . . . 7
- 2.2 Bonuses, Penalties & Grade Calculation . . . 10
  - 2.2.1 Example Grading Session . . . 11
- 2.3 Constructing `pretor.toml` Files . . . 14
  - 2.3.1 A Sample `pretor.toml` . . . 14
  - 2.3.2 Bypassing Metadata Checks . . . 14
- 2.4 Constructing Course Definitions . . . 15
  - 2.4.1 A Sample Course Definition File . . . 16
- 2.5 Advanced Grading REPL Topics . . . 17
  - 2.5.1 The Symbol Table . . . 17
  - 2.5.2 Writing a RC File . . . 18
- 2.6 Exporting Grades . . . 18
  - 2.6.1 Sample Usage of `pretor-export` . . . 18

**3 The Pretor Submission File (PSF) Format** **21**
- 3.1 Pretor Submission File Format Revisions . . . 21
- 3.2 File Structure . . . 21
  - 3.2.1 Understanding Revisions . . . 22
  - 3.2.2 The `pretor_data` Schema . . . 22
  - 3.2.3 The `rev_data` Schema . . . 22
- 3.3 Forensic Information . . . 23

**4 Pretor for Systems Administrators** **25**
- 4.1 Deploying Pretor with `dpkg` . . . 25
- 4.2 Deploying Pretor with `pip` . . . 25
- 4.3 Deploying Pretor with `pyinstaller` . . . 25
- 4.4 Security Considerations . . . 25
- 4.5 System Requirements . . . 26
- 4.6 Licensing . . . 26

# 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

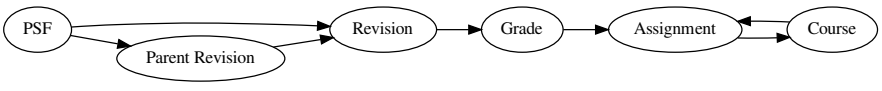


Figure 1.1: High-level overview of the Pretor data model

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

When a student generates a PSF from a project directory, `pretor-psf` uses the `pretor.toml` file in the top-level directory of the project to "burn in" metadata such as the assignment, course, semester, and section number. While this information can also be supplied using command-line arguments[3], the use of `pretor.toml` reduces the potential for human error which is important given that metadata fields are often matched using exact string comparison.

Typically, the instructor for a course will write a `pretor.toml` file for each assignment (and possibly for each section[4]. The `pretor.toml` file may either be provided to students with an assignment template, or by asking students to download and install it into their projects.

A `pretor.toml` file may define any of the following fields:

- `exclude` – a list of glob patterns to exclude from being included in the generated PSF
- `course` – the string name of the course, i.e. "CS101"
- `section` – the section identifier[5]
- `semester` – the string name of the semester, i.e. "Spring 1994"
- `assignment` – the string name of the assignment, i.e. "Assignment 1"
- `minimum_version` – the minimum Pretor version which can be used to pack this assignment as a string, i.e. "0.0.2"

### 2.3.1 A Sample `pretor.toml`

```
assignment = "Assignment 1"
section = 2
semester = "Spring 1973"
course = "ABC123"
exclude = ["*.o"]
```

### 2.3.2 Bypassing Metadata Checks

**Danger Zone**: The information in this section will allow you to generate PSFs with missing or incorrect metadata which may impede grading.

**Students Beware**: As this documentation is distributed publicly, it is entirely possible that student users of Pretor may stumble upon this information. The procedures described here are deliberately hidden from the "help" information of `pretor-psf` to protect you from "shooting yourself in the foot". Using these procedures has the potential to make your submissions significantly more difficult

[3] see `pretor-psf --help`

[4] According to the needs of the specific course, the section number/identifier may be provided via `pretor.toml` or via `pretor-psf --section`

[5] Note that the section identifier does not have to be numeric, although it is occasionally referred to as such. The section identifier is stored as a string to avoid constraining University section numbering conventions

> to grade for your instructor, which is unlikely to endear you to them. Consider yourself warned.

Pretor features three checks to help prevent user error on the part of students while packing assignments to PSF for submission. These may be disabled via (intentionally) undocumented options to `pretor-psf`. In some cases, it may be necessary for an instructor, developer, or administrator to bypass one or more of these checks. The correct procedures to do so are documented below.

**Metadata Check** This check requires that all of the metadata fields "semester", "section", "assignment", "group", and "course" are specified either via command-line arguments, or via `pretor.toml`. It may be disabled via the flag `--no_meta_check`. Specifying this flag will assert the key `no_meta_check` in both the forensic information and metadata of the generated PSF.

**`pretor.toml` Check** This check requires that the top level directory of the submission directory contains a `pretor.toml` file. This is intended to catch cases of users accidentally specifying the wrong directory to generate a PSF from. This check may be disabled using `--allow_no_toml`, doing so will assert the key `allow_no_toml` in both the forensic information and metadata of the generated PSF.

**Version Check** This check allows an instructor to specify a minimum Pretor version to be used for packing the assignment via the `minimum_version` key of `pretor.toml`. This is so that if a future Pretor version adds a new feature that is needed to correctly pack the submission, student users will not accidentally use an outdated version. This check may be bypassed using the flag `--disable_version_check`, which will assert the field `disable_version_check` in both the forensic information and metadata of the generated PSF.

## 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[6]. 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.

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

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

### 2.4.1 A Sample Course Definition File

```
[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"
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
```

## 2.5 Advanced Grading REPL Topics

> **Note**: The full array of commands available in the grading REPL is not documented here, you can use `help` to view a list of commands or `help <command>` to view documentation for a specific command.

The Pretor grading REPL (accessed via `pretor-grade`) is in fact a robust domain-specific language for interacting with Pretor's internal data structures. Although it is intentionally not Turing-complete[7], the grading REPL contains many useful features including:

- A user readable and writable symbol table for handling runtime configuration (an entry in the symbol table is conceptually equivalent to a variable).
- A convenient method to view the REPL's internal state – via rad-only entries in the symbol table.
- History (via the `history` command) and tab completion.
- Capability to execute shell commands by prefixing them with `!`, i.e. `!echo hello`.
- Basic error handling and recovery facilities.

### 2.5.1 The Symbol Table

The symbol table can be directly interacted with via three main commands:

- `symtab` – display all symbols in the symbol table and their current values. Note that symbols prefixed with `#` are read-only, and are generally used internally for book-keeping by the REPL.
- `get` – get the value of a single symbol (i.e. `get symname`).
- `set` – set the value of a single symbol (i.e. `set symname newval`). The old value is returned as the result of the command.

The symbols in the symbol table are documented in figure 2.2. Note that additional symbols may be added by custom RC scripts or via plugins.

[7] It is in fact possible that the grading REPL is Turing-complete, but this has been deliberately not investigated, as it is not intended to be. Don't turn your grading workflow into a Turing tarpit. You have been warned

| Symbol | Purpose |
|---|---|
| `#result` | The result of the most recently executed command, this is displayed to the console after the command finishes running. |
| `#lastresult` | The result of the second most recently executed command. |
| `#status` | True if the command finished successfully, otherwise False. |
| `#lastatatus` | The status for the previous command. |
| `#error` | The error text if an error occurred by the command. |
| `#finalized` | A list of indices of #psf which have been finalized already |
| `#argv` | Split list of arguments to the current command. |
| `#psf` | A list of currently loaded PSF objects. |
| `coursepath` | :-delimited list of directories to search for course definition files in. |
| `outputdir` | The directory where finalized PSFs are written to. |
| `revision` | The revision to interact with when using the `interact` command. If empty, an appropriate value is generated automatically. This may be overridden to inspect a specific revision in a PSF. |
| `base_revision` | The revision to use as the base ungraded revision submitted by the student. This usually does not need to be modified |

Figure 2.2: Table of symbols and their purposes.

### 2.5.2 Writing a RC File

By default, all commands stored in `~/.config/pretor/rc` are executed in order when the REPL starts up. The file used for this purpose may be overridden via `pretor-grade --rc`. Lines beginning with the # character are ignored to facilitate leaving comments.

This file is useful for setting up configuration options that will be used on a regular basis. For example, the course definition search path, or finalized PSF output directory might be defined here to avoid needing to specify these for each grading session.

## 2.6 Exporting Grades

Grades may be exported for upload to a LMS[8]. This is accomplished via the `pretor-export` command. This command iterates over a collection of PSF files and generates output in one of several formats. For full documentation, see `pretor-export --help`.

The Moodle formatted output[9], which is simply a text CSV format, is likely the most amenable to the application of your own automation, and should be well suited for parsing with tools such as Excel, LibreOffice, or xsv[10].

### 2.6.1 Sample Usage of `pretor-export`

```
$ pretor-export --input *.psf --moodle
"Spring 1973","ABC123",2,"cad",100.0,""
$ pretor-export --input *.psf --table
Spring 1973  ABC123  2  cad  100
$ pretor-psf --scorecard --input Spring\ 1973-ABC123-2-cad-Assignment\ 1.psf
SCORECARD FOR ABC123: Assignment 1

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

[8] Learning Management System, such as Moodle, Blackboard, et. al.

[9] Obtained via the `--moodle` flags

[10] `https://github.com/BurntSushi/xsv`

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

OVERALL SCORE: 100.00%
```

# Chapter 3

# The Pretor Submission File (PSF) Format

## 3.1 Pretor Submission File Format Revisions

PSF includes in it a format revision number, which is in place to allow future versions of Pretor to detect files created by older versions. The history of each version is documented here.

| format revision | introduced | deprecated | Changes |
|---|---|---|---|
| 0 | 0.0.1 | cur. | Initial PSF format revision. |

Figure 3.1: History PSF format revisions

## 3.2 File Structure

All PSFs are valid zip files[1], but use the `.psf` file extension for clarity. The metadata and other information pertaining to a given PSF is stored as plain files in the zip, which are enumerated in figure 3.2.

[1] See also the PKZIP Application Note: `https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT`

| Path | Purpose |
|---|---|
| `/pretor_version` | Plain-text file containing the Pretor version string of the Pretor instance which created this file. |
| `/psf_format_revision` | Plain-text file containing the integer PSF Format Revision as a string. |
| `/pretor_data.toml` | A TOML formatted file containing various data about the PSF, see §3.2.2. |
| `/revisions/` | Directory containing information about revisions in this PSF, see §3.2.1. |
| `/revisions/*/rev_data.toml` | A TOML formatted file containing information about a given revision. See §3.2.3. |
| `/revisions/*/grade.toml` | A TOML formatted file containing the grade for a given revision, see § 2.1 and §2.2. This file is optional. |
| `/revisions/*/course.toml` | A TOML formatted file containing the course definition associated with a given revision's grade, see §2.4. If `grade.toml` is present, this file must also be present, and vice-versa. |
| `/revisions/*/contents/` | This directory contains the files which are associated with the revision. It may contain any arbitrary file structure as is desired. |

Figure 3.2: Table showing the purpose of each file within a PSF formatted zip. `/` is assumed to be the top-level of the zip.

### 3.2.1 Understanding Revisions

The PSF format and associated data model support usage as a system for tracking arbitrary revisions, although none of the user interfaces provided by Pretor permit this directly. Instead, the chain of revisions (similar to git commits) is always kept linear, beginning with a base revision (generally `submission` created by `pretor-psf`), with an intervening chain of grade revisions (named by the pattern `grade_[0-9]+` ascending).

The purpose of this system is to provide an easily-audit-able record of the exact data the student turned in, as well as their grade and every revision made to that grade. Internally, a revision is simply a directory within the zip file storing normal files. At time of writing, Pretor supports neither diff-ing revisions, nor storing only files changed between revisions.

### 3.2.2 The `pretor_data` Schema

The `pretor_data.toml` file must contain the following keys:

- `pretor_version` – the version string of the Pretor instance which packed this PSF.
- `ID` – UUID of this PSF. At time of writing, the ID field is unused, but in the future it may be used as a primary key for differentiating PSFs.
- `revisions` – a list of revision IDs which this PSF contains. Only revisions in this list will be considered, even if the relevant directories exist in `revisions/`.

The `pretor_data.toml` file may optionally contain the `metadata` key, which may be used to store arbitrary metadata. Although the underlying PSF implementation imposes no special restrictions on the schema of this field, other portions of Pretor assume that this is a key-value-pair store in the form of a dictionary, usually a superset of the information provided in `pretor.toml` in the original directory used to generate the PSF.

### 3.2.3 The `rev_data` Schema

The `rev_data.toml` must contain the `ID` field (which stores the revision ID as as a string), and the `contents` field, which stores a list of files (as relative paths from the relevant `contents/` directory). It may also

optionally contain a `parentID` field, which is the revision ID of the parent revision, if any; an omission of this field implies that this revision has no parent.

## 3.3 Forensic Information

> **Note**: The forensic data stored by Pretor is not encrypted, or is it signed. Any attacker with an understanding of the Pretor source code who has ever possessed a given PSF could modify the forensic data in arbitrary ways.

Every PSF contains forensic metadata which is burned into the zip in a fashion which is deliberately not documented. The manner in which this data is attached to the zip file is unrelated to the normal storing of file and directory entries within the zip. This is intended to keep honest students honest by recording various information about who packed the PSF and on what machine. It is assumed that the lack of accessible documentation regarding this data will act as it's own deterrent, in that students who are capable of sussing out the means by which the forensic data is stored from the source code would have no reason to tamper with it. A PSF with missing forensic information is still perfectly valid, though first-party Pretor tooling will throw warnings if it is missing.

The forensic data stored within a PSF may be viewed using the `forensic` command in `pretor-grade`, or via `pretor-psf --forensic`.

If an instructor has reason to believe that a student submission may have been tampered with (such as altering the creation timestamp or group ID), then they are encouraged to inspect the forensic data of any relevant PSFs. Forensic information containing information which is inconsistent with the metadata of the same file suggests that the PSFs contents have been tampered with[2].

[2]Note that the timestamp stored in the forensic data by `pretor-psf` is generated separately from the timestamp stored in the metadata, it is normal for the two to differ by up to several seconds.

# Chapter 4

# Pretor for Systems Administrators

The student-facing component of Pretor consists only of the `pretor-psf` command. In a lab environment where instructors will be grading on a different set of machines, only this component of Pretor needs to be installed. However, it is often easier to simply install the entirety of the package. This chapter documents several approaches to accomplish this.

At time of writing, Pretor does not have any system-wide configuration files which need to be managed, although you may wish to provide a "default" RC file (see §2.5.2) for graders to use. At this time, the only way to accomplish this is by placing said file in the user's home directory.

Additionally, you may wish to place all course definition files which instructors in your environment may be using in a convenient central location, such as `/etc`. Be aware however that your instructors will likely need to make tweaks to these files throughout each semester.

## 4.1 Deploying Pretor with `dpkg`

Via the `python3-stdeb` package, Pretor (or any other setuptools based Python package) may be used to generate a dpkg-compatible `.deb` file. Beginning with 0.0.3, regular Pretor releases include a pre-built `.deb` file. Such a file may be generated by the command: `python3 setup.py -command-packages=stdeb.command bdist_deb`. The relevant binary file will be created in `deb_dist/`.

At time of writing, this is the suggested approach for production installations of Pretor.

## 4.2 Deploying Pretor with `pip`

Pretor uses `setuptools` and may be installed as a standard Python package via `python3 setup.py install`.

## 4.3 Deploying Pretor with `pyinstaller`

At present, only `pretor-psf` may be distributed as a pyinstaller-created binary. This is to facilitate easy distribution to students without needing to account for dependencies or other considerations.

For those Pretor components which support distribution in this fashion, wrapper scripts are provided in the `pyinstaller/` directory of the Pretor source distribution. Each can be used to generate a binary via the `pyinstaller -onefile pyinstaller/<filename>.py` command. Generated binaries are placed in `dist/`.

Regular Pretor releases to not include pyinstaller-based binary builds.

## 4.4 Security Considerations

Pretor (in particular `pretor-grade`) inherently involves consuming arbitrary files provided by students and executing code stored within them. This is necessary for the purpose of grading computer science programming assignments. Precautions are taken to reduce the changes of a maliciously constructed input

causing damage to the host system, but ultimately the PSF de-serialization methods rely on Python's ZIP and TOML implementations. To that end, security advisories relating to Python or it's ZIP or TOML packages will also relate to Pretor.

Be aware also that `pretor-grade` is not indented to "contain" it's user. The interpreter features shell escapes, and a standard and commonly used interpreter command spawns a Bash instance. You should not allow people to run `pretor-grade` on systems that you do not wish them to have shell access to.

## 4.5 System Requirements

Pretor officially supports Python 3.5, 3.6, and 3.7 running on Ubuntu 16.04 or Ubuntu 18.04. Pretor depends on language features not implemented until Python 3.5, namely pathlib and type hinting. It is unlikely to work in older Python versions, though it may with appropriate back porting. Versions of Python older than 3.5 are not, and never will be officially supported. New Python releases will be tracked as they become available, and you can expect Pretor to continue working with new Python releases for as long as Pretor continues to be maintained.

Pretor should be "well behaved", in that it does not generally make dangerous assumptions such as "paths are strings". It is expected to work on any system, including Windows, where appropriate Python versions are available, although bear in mind that Bash is a requirement for grading. Pretor is not actively tested on systems other than those noted, and support cannot be guaranteed.

## 4.6 Licensing

> **Disclaimer**: nothing in this section is legal advice, nor is anything in this section a part of the Pretor license. This section is provided only to act as a reminder of Pretor's license and your responsibilities stemming therefrom. This section is not an exhaustive list or summary of responsibilities, rights, or any other information pertaining to or contained in the Pretor license. In any case where content in this section might conflict with the Pretor license through author error or otherwise, the Pretor license takes precedence.

Pretor is AGPL licensed, as noted prominently in the README of the source distribution. You should become familiar with the terms of this license if you are not already. Keep in mind that while Pretor may be used as an "application server", such as for batch-processing of grades, all users of any such service are entitled to it's source code.

While development in may forms, such as forks, upstream contributions, plugins, or by using Pretor as a library are all encouraged, Pretor is not intended to be used as a commercial or for-profit product. The AGPL license was specifically selected to prevent individuals or other legal entities from creating closed-source forks or distributions of Pretor.

To clarify one potential gray area, the Pretor authors do not consider work submitted by students via Pretor to be derivative works, and are consequentially exempt for the Pretor license. In other words, the use of Pretor in an academic environment has no effect on the licensing or ownership of student code, i nthe view of Pretor's authors.