Розділ 7
Публікація вашого першого проєкту Pharo

У цьому розділі ми детальніше пояснимо, як ви можете за допомогою Iceberg опубліку-вати свій проєкт на GitHub. Iceberg – це інструмент і бібліотека, що дає змогу керувати Git-сховищами. Він робить більше, ніж просто зберігає та публікує ваші файли. Тоді ми коротко покажемо, як переконатися, що і ви можете, й інші розробники можуть завантажити ваш код, не розуміючи його внутрішніх залежностей.

Зауважимо, що цей розділ написано для незалежного читання, тому можуть трапляти-ся повторення матеріалу інших розділів.

Ми не будемо пояснювати таких базових понять як *commit* (запис до сховища), *push* (вивантаження до сховища), *pull* (завантаження зі сховища), чи клонування сховища. За потреби зверніться до посібника з Git. Для читання цього розділу важливо, щоб ви мали можливість з командного рядка публікувати файли на вашому віддаленому сервісі Git. Якщо у вас немає такої можливості, то не сподівайтеся, що Iceberg надасть її. Якщо у вас виникли проблеми з конфігурацією протоколу SSH (який є способом за замовчуванням вивантаження на GitHub), то можете використати замість нього HTTPS, або прочитати книжечку «Manage your code with Iceberg», яку можна знайти на http://books.pharo.org. Давайте почнемо.

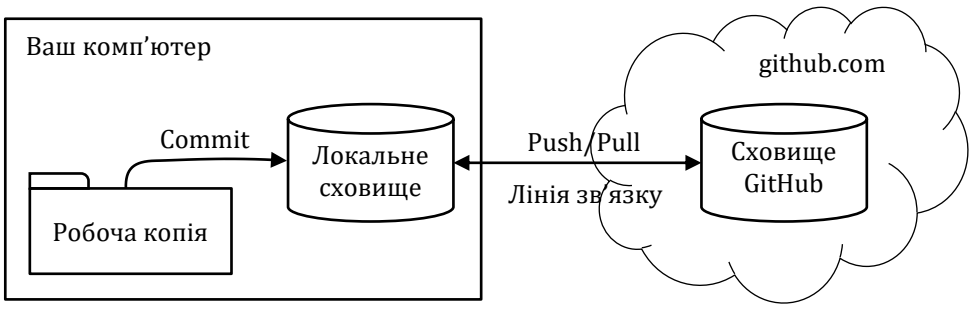

Рис. 7.1. Git – розподілена система контролю версій

7.1. Для нетерплячих

Якщо ви не хочете читати все написане нижче, і відчуваєте себе досить впевнено, ось стислий перелік кроків для того, щоб опублікувати свій код:

- створіть сховище на GitHub або будь-якій іншій Git-платформі;
- [не обов'язково] налаштуйте Iceberg на використання спеціальних ключів SSH;
- додайте проєкт до Iceberg;
- [не обов'язково, але дуже рекомендовано] створіть у вашому локальному сховищі папку з іменем «src» – це правильна загальна домовленість;
- відкрийте свій проєкт у Iceberg і додайте до нього пакети (класів);
- запишіть свій проєкт (виконайте *commit*);

- [не обов’язково] додайте до проєкту базові параметри;
- вивантажте зміни до віддаленого сховища.

І ви закінчили! А тепер давайте пояснимо ці кроки спокійніше.

7.2. Базова архітектура

Оскільки Git є розподіленою системою контролю версій, то вам потрібно мати локальний клон віддаленого сховища та робочу копію проєкту. Локальне сховище та робоча копія зазвичай розташовуються на вашій машині. Зроблені вами зміни в робочій копії будуть записані до локального сховища перед вивантаженням до віддаленого сховища чи сховищ (див. рис. 7.1). Ми використовуємо Iceberg тому, що Pharo трохи ускладнює процес зберігання. Пояснимо у двох словах, чим саме: класи та методи Pharo – це об’єкти, які змінюються на льоту. Коли ви змінюєте вихідний код класу, робоча копія Git не змінюється автоматично. У вас ніби є дві робочі копії: одна – об’єктна в образі вашої системи, інша – на основі Git-файлу. Iceberg створено для того, щоб допомогти синхронізувати їх та керувати ними обома.

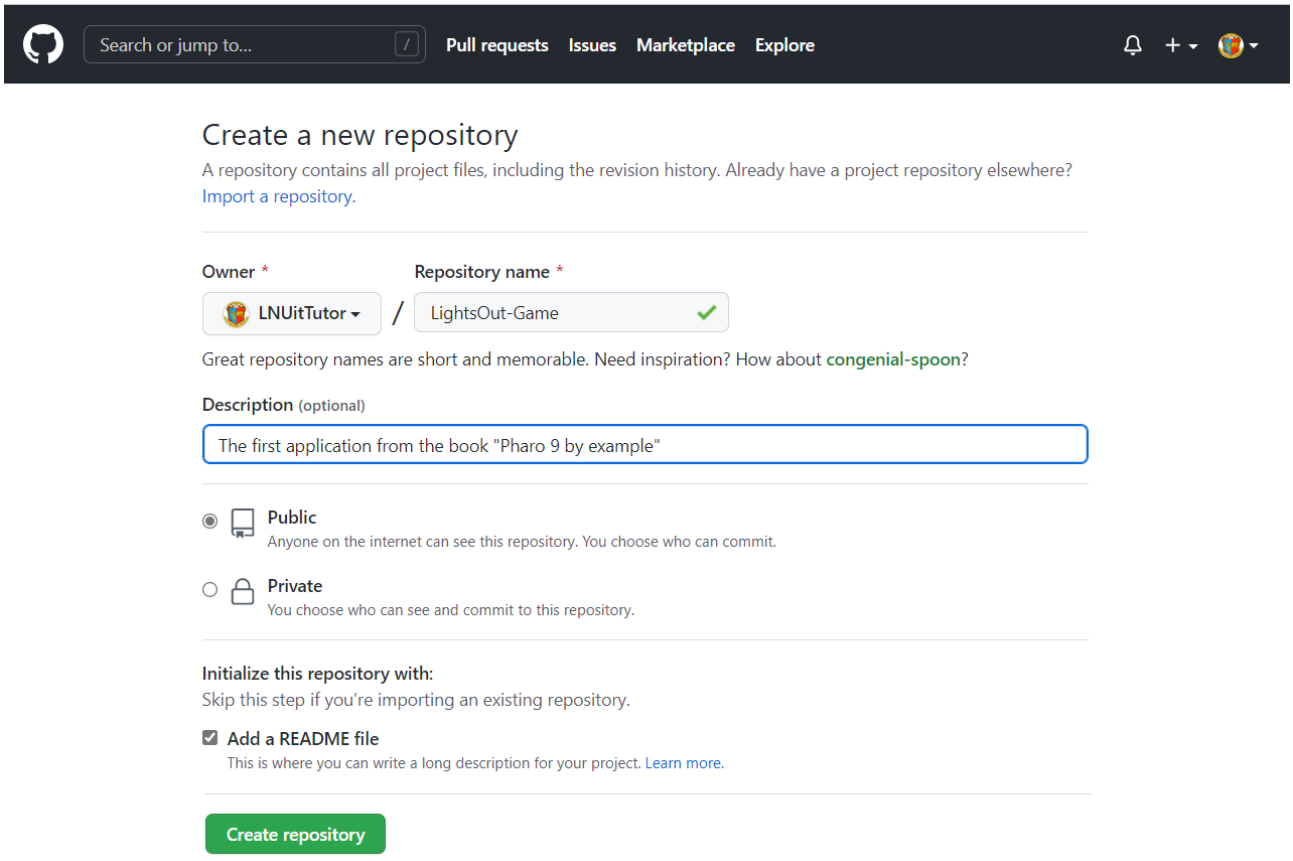

Рис. 7.2. Створення нового сховища на GitHub

Створення нового сховища на GitHub

Вам потрібно створити два сховища: локальне та віддалене. У розділі 5 ми починали з локального, а тепер покажемо, як спершу створити репозиторій на GitHub. Черговість не має особливого значення, але, якщо ви почали з GitHub, то доведеться виконати додаткові кроки у Pharo, про які зараз поговоримо.

Під час створення сховища на GitHub потрібно задати його ім’я, опис і додати файл *README* (див. рис. 7.2).

Налаштування SSH: повідомте Iceberg свої ключі

Щоб мати змогу зберігати код у сховищі на GitHub, ви повинні використати для доступу або HTTPS, або SSH. У першому випадку, як уже було сказано в п. 5.13, вам буде потрібен персональний токен доступу, у другому – потрібно буде налаштувати дійсні облікові дані у вашій системі. Якщо ви використовуєте SSH (спосіб за замовчуванням), то вам потрібно буде переконатися, що ці ключі доступні для вашого облікового запису GitHub, а операційна система додала їх до клієнта SSH для зв'язку з сервером.

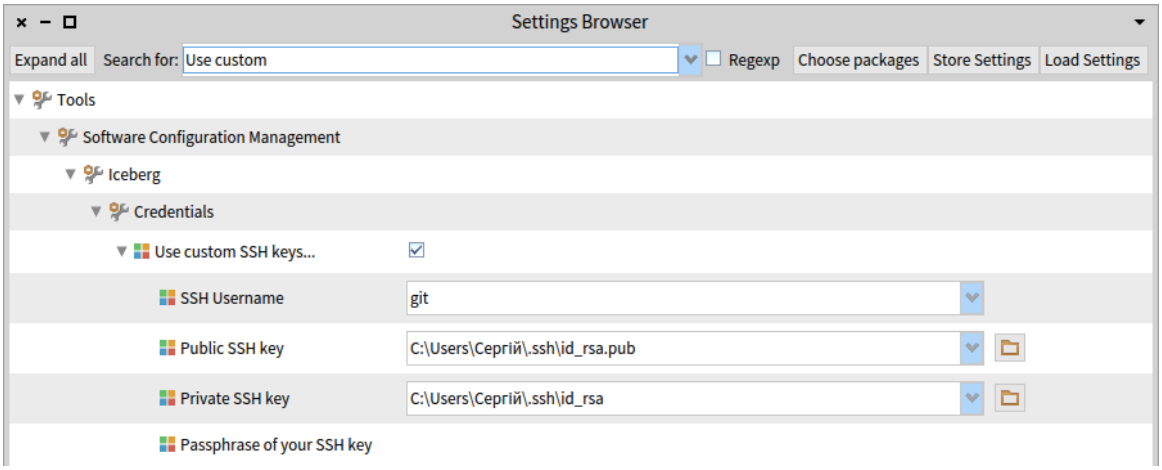

Рис. 7.3 Налаштування ключів SSH

Відкрийте оглядач налаштувань командою меню «*Pharo* > *Settings*» і введіть у рядку пошуку «*Use custom SSH keys*» – так ви швидко перейдете до потрібних налаштувань, що розташовані десь наприкінці всього списку. Введіть посилання на файли з вашими ключами, як показано на рис. 7.3.

Налаштування можна також виконати програмно: виконайте в Робочому вікні такий фрагмент коду

```
IceCredentialsProvider useCustomSsh: true.
IceCredentialsProvider sshCredentials
   publicKey: 'path\to\ssh\id_rsa.pub';
   privateKey: 'path\to\ssh\id_rsa'
```

Зауваження. Для зберігання ключів можна використовувати файли з нестандартними іменами. У цьому випадку потрібно замінити «*id_rsa*» в коді (чи в налаштуваннях) іменем вашого файлу.

Тепер ми готові поглянути на те, як Iceberg керує Git у Pharo.

Від перекладача. Операційна система Windows 10 уже містить готового клієнта SSH. Вам доведеться лише згенерувати ключі та додати їх до клієнта в режимі командного рядка. Докладний опис усіх кроків щодо створення ключів та приєднання їх до облікового запису GitHub можна знайти в посібнику за адресою https://docs.github.com/en/authentication/connecting-to-github-with-ssh, а опис виправлення поширених помилок – https://docs.github.com/en/authentication/troubleshooting-ssh.

Якщо вам не вдалося налаштувати доступ через клієнта SSH, то не переживайте, продовжуйте використовувати HTTPS – він чудово працює.

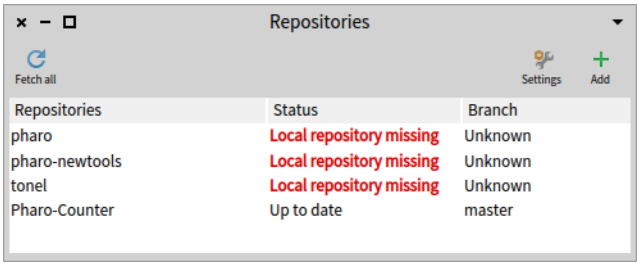

Рис. 7.4. Оглядач репозиторіїв Iceberg, відкритий в новому образі системи

7.3. Про оглядач репозиторіїв Iceberg

На рис. 7.4 зображено головне вікно Iceberg. Воно містить перелік відомих проєктів (сховищ). Серед них є створений нами у розділі 5 *Pharo-Counter* і кілька проєктів, отриманих з образом Pharo під час встановлення системи, наприклад, *pharo*. Напис червоного кольору «*Local repository missing*» означає, що Iceberg не може знайти відповідне локальне сховище.

Передусім не хвилюйтеся про відсутній репозиторій для Pharo: він вам знадобиться, лише якщо захочете зробити внесок у розробку мови Pharo. Звичайно, ми усіх запрошуємо долучатися до проєкту, але, напевне, давайте спочатку закінчимо цей розділ. Ось що відбувається: Iceberg попереджає, що система Pharo не знає, де розміщений відповідний репозиторій Git для її класів. Але ви вільно можете переглядати системні класи й методи, вносити зміни, Pharo чудово працює без локального сховища, бо має вбудовану систему керування версіями коду. Це попередження свідчить лише про те, що якщо ви хочете керувати версіями системного коду Pharo за допомогою Git, то ви мусите повідомити системі, де на вашій локальній машині збережені локальний клон і робоча копія Pharo. Якщо ви не плануєте змінювати системний код Pharo та ділитися ним, то не варто про це турбуватися.

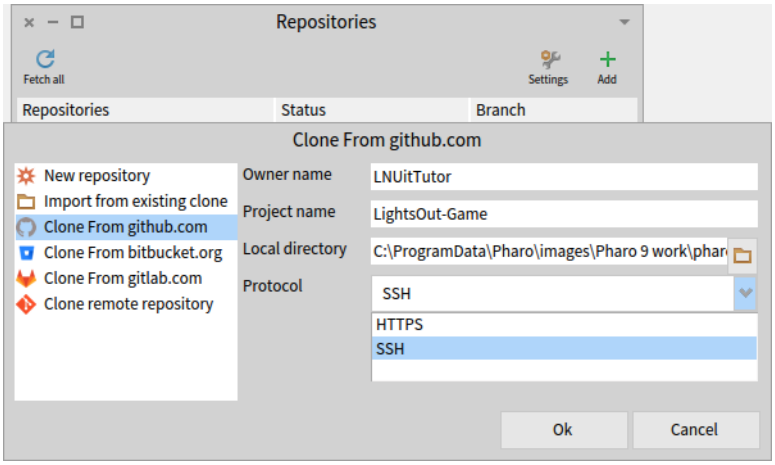

Рис. 7.5. Клонування проєкту з GitHub через SSH або HTTPS

7.4. Додавання нового проєкту до Iceberg

Спочатку додайте проєкт до Iceberg.

- Натисніть кнопку ***Add***, розташовану вгорі праворуч у головному вікні Iceberg.
- Вкажіть розташування репозиторію. Ми копіюємо репозиторій з GitHub, тому потрібно вибрати відповідну опцію, вказати ім'я власника і назву репозиторію.

Нагадуємо, що ви можете використати один з протоколів: SSH або HTTPS (рис. 7.5).

Задані параметри вказують Iceberg клонувати репозиторій, який ми щойно створили на GitHub. Ми задали власника, ім'я сховища, шлях до папки на диску свого комп'ютера, де розташуються локальний клон і робоча копія сховища.

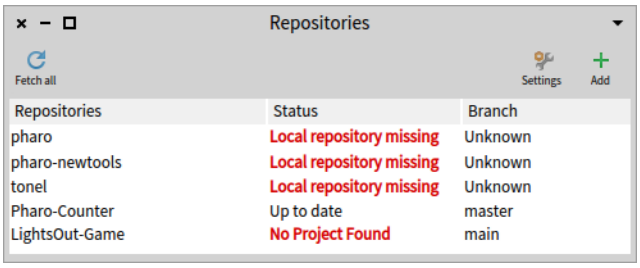

Рис. 7.6. Одразу після клонування порожнього репозиторію Iceberg повідомляє, що в проєкті бракує інформації

Тепер Iceberg додасть заданий репозиторій до списку керованих проєктів і клонує порожній репозиторій на локальний диск. Ви мали б побачити щось схоже на рис. 7.6. Закономірно, що проєкт має статус «*No Project Found*», адже він порожній, і Iceberg не може знайти його метадані. Зараз виправимо цю помилку.

Згодом, коли ви збережете зміни до свого проєкту і захочете завантажити його до іншого образу, клонувавши його ще раз, ви побачите, що Iceberg повідомить інший статус: «*Not loaded*» – проєкт є з усіма даними, проте ще не став частиною образу.

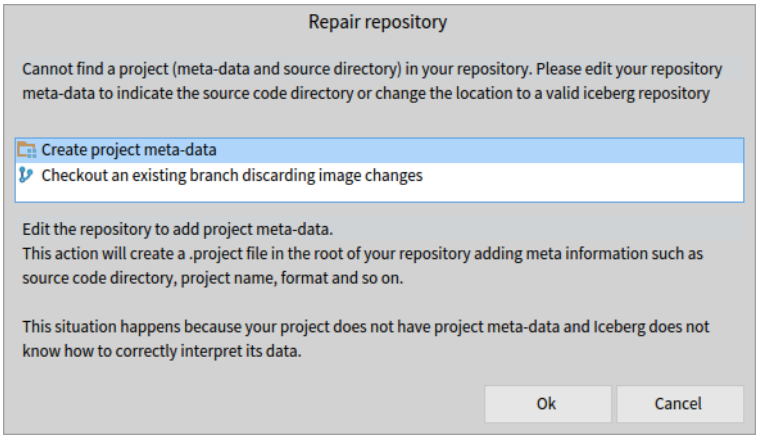

Рис. 7.7. Дія створення метаданих проєкту та її пояснення

Рятівне відновлення

Iceberg – це досить розумний інструмент, здатний допомогти вам вирішити проблеми, які можуть трапитися під час роботи з Git. Загалом, коли ви бачите статус репозиторію, відображений червоним (наприклад, «*No Project Found*», або «*Detached Working Copy*»), попросіть Iceberg виправити ситуацію за допомогою команди «*Repair repository*» контекстного меню проєкту.

Iceberg не може виправити всі помилки автоматично, натомість він запропонує вам можливі шляхи вирішення та пояснить кожен з них. Можливі дії буде впорядковано від імовірно більш правильних до менш. Кожну дію супроводжує пояснення, що вона робить, та як її застосовувати. Завжди варто читати такі пояснення. Якщо правильно налаштувати репозиторій, то дуже важко втратити будь-який фрагмент коду з Iceberg і Pharo, оскільки Pharo зберігає власну копію коду. Дуже важко, проте… можливо. Тому будьте уважні!

Створення метаданих проєкту

Iceberg повідомляє, що він не може знайти проєкт, оскільки немає деяких його метаданих, такі як спосіб кодування файлу та приклад розташування всередині репозиторію. Коли ми запустимо команду відновлення сховища, то побачимо нове вікно, зображене на рис. 7.7. Оберіть у ньому дію «*Create project metadata*» та прочитайте пояснення.

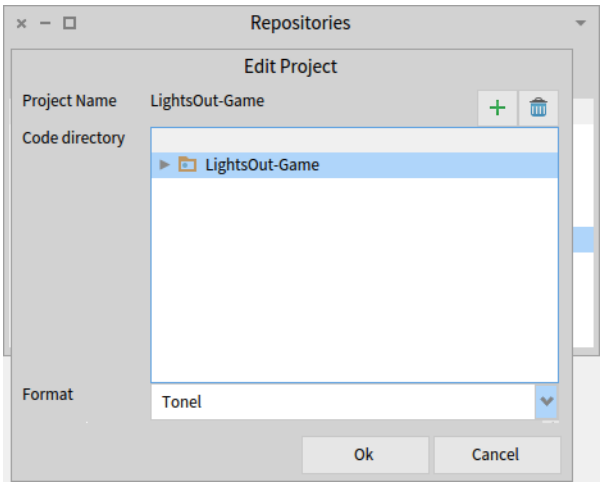

Рис. 7.8. Місце зберігання метаданих і формат проєкту

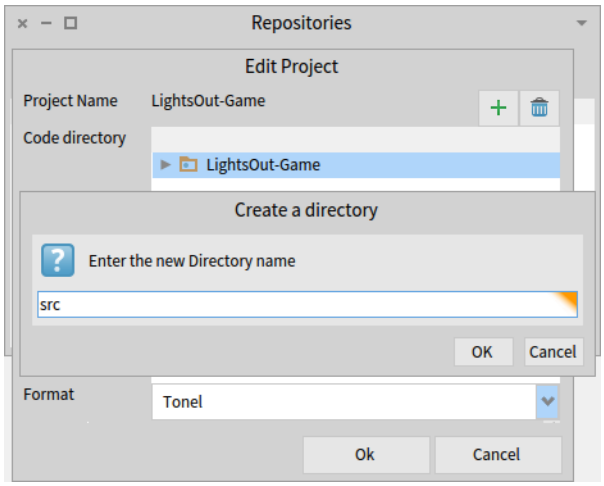

Рис. 7.9. Додавання папки для зберігання коду

Коли ви підтвердите свій вибір, Iceberg покаже структуру папок вашого проєкту та його формат, як на рис. 7.8. Бажаним форматом проєктів Pharo є *Tonel*, бо його розроблено спеціально для використання в різних файлових системах. Формат *Tonel* можна змінити на *Filetree*, проте робіть так лише тоді, коли воно вам справді потрібно.

Ще один не зайвий крок – додати до проєкту папку для зберігання коду. За домовленістю її називають «*src*». Натисніть кнопку з зеленим знаком плюс і введіть ім'я папки у вікні діалогу, як показано на рис. 7.9. Сформовану структуру папок проєкту показано на рис. 7.10. Не забудьте позначити новостворену папку – тільки так ви справді вкажете, що код потрібно зберігати в «*src*». Тисніть ще раз ***Ok***, і відновлення метаданих буде завершено. Назву проєкту позначено зірочкою та зеленим кольором (див. рис. 7.11), бо він містить незбережені зміни. Виконайте команду «*Commit*» контекстного меню проєкту. Iceberg покаже деталі операції (рис. 7.12), а сам проєкт набуде вигляду, як на рис. 7.13.

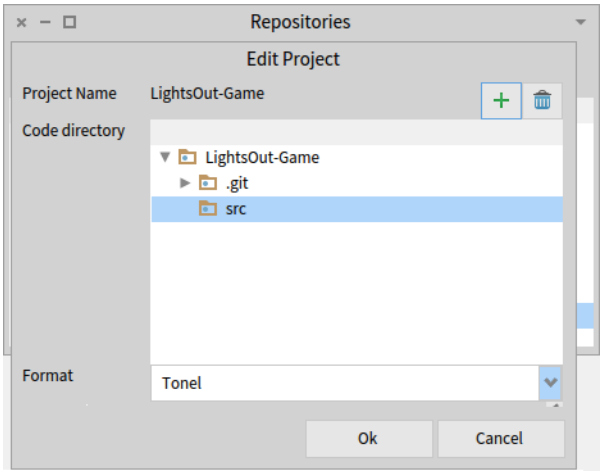

Рис. 7.10. Сформована структура проєкту

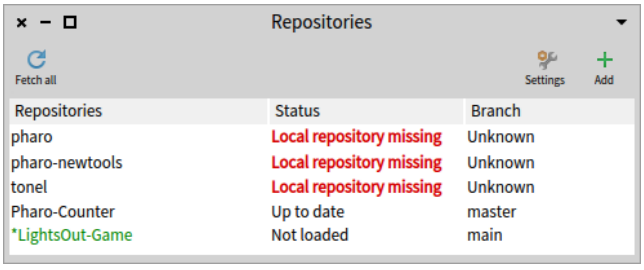

Рис. 7.11. Після відновлення метаданих проєкт містить незбережені зміни

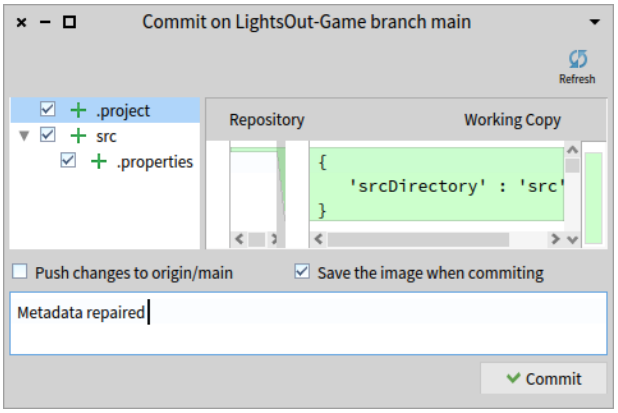

Рис. 7.12. Деталі збереження метаданих

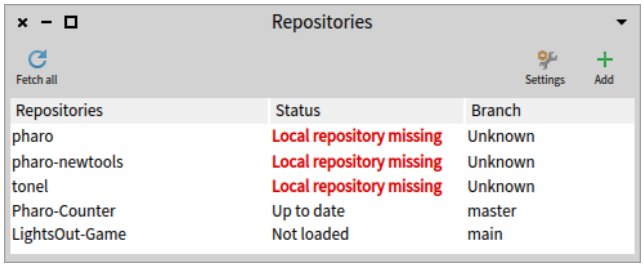

Рис. 7.13. Після відновлення метаданих проєкт містить незбережені зміни

Після того, як ви збережете метадані, Iceberg покаже вам, що проєкт відновлено, але не завантажено, як показано на рис. 7.13. Це нормально, оскільки ми ще не додали до проєкту жодних пакетів. Далі, якщо хочете, можете надіслати збережені зміни до свого віддаленого сховища за допомогою команди «*Push*» контекстного меню проєкту.

Ваше локальне сховище готове, давайте перейдемо до наступної частини.

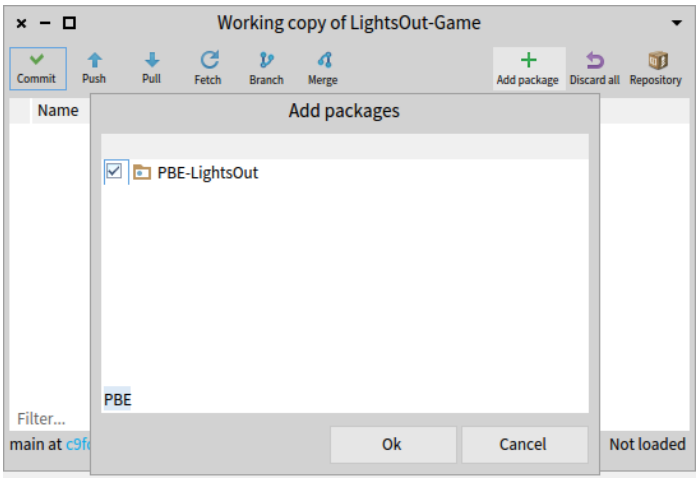

Рис. 7.14. Додавання пакета за допомогою оглядача робочої копії

7.5. Додайте і збережіть пакет за допомогою оглядача робочої копії

Оскільки ваш проєкт уже містить метадані, то тепер Iceberg легко зможе керувати ним. Двічі клацніть на імені проєкту, щоб відкрити його в оглядачі робочої копії. Оглядач відображає перелік пакетів, з яких складається проєкт. У вас він поки що порожній. Додайте пакет «*PBE-LightsOut*» за допомогою кнопки ***Add package***, як показано на рис. 7.14. Iceberg знову просигналізує зеленим кольором і зірочкою перед іменем проєкту про те, що він містить незбережені зміни (див. рис. 7.15).

Збережіть зміни

Збережіть зміни до локального сховища, клацнувши кнопку ***Commit***, як зображено на рис. 7.15. Iceberg демонструє всі частини коду: класи й методи, – які зазнали змін. За замовчуванням усі вони позначені для зберігання. За потреби ви можете змінити ці позначки на власний розсуд: це необов'язкова, проте важлива можливість. Після збереження Iceberg змінить колір імені проєкту на чорний та опис стану на «*1 not published*». Це означає, що образ системи та робочу копію синхронізовано з локальним сховищем,

але воно відрізняється від віддаленого: зміни ще не опубліковано на GitHub. За потреби ви можете зберігати зміни до локального середовища кілька разів підряд.

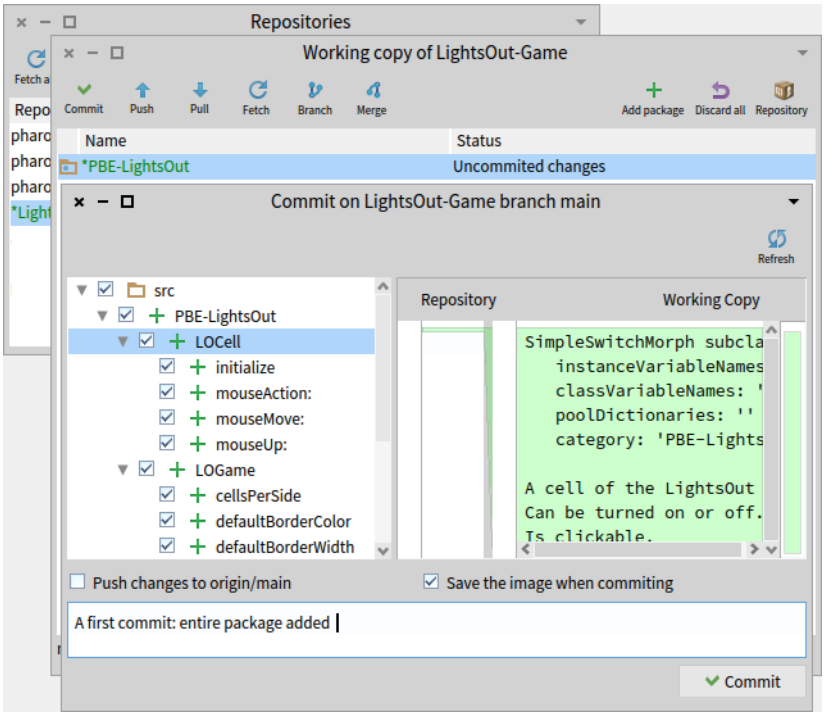

Рис. 7.15. Iceberg та оглядач робочої копії сигналізують, що в проєкті є незбережені зміни – доданий нами пакет. Під час виконання команди *Commit* ви можете вибрати, які саме частини коду зберегти

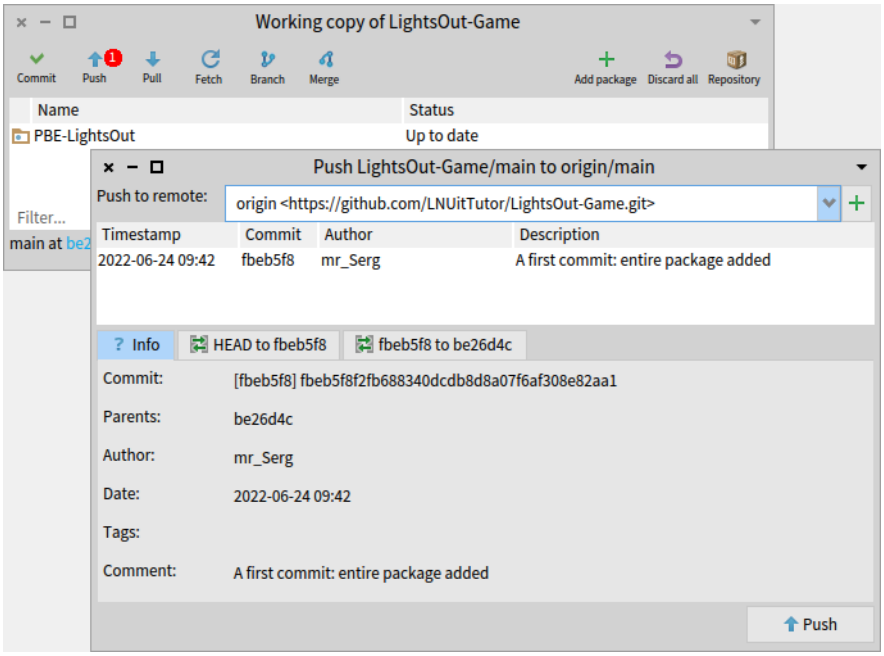

Рис. 7.16. Вивантаження збережених змін

Вивантажте зміни до віддаленого сховища

Натисніть кнопку ***Push*** оглядача робочої копії, щоб перенести збережені зміни з локального сховища до віддаленого. Кнопку легко помітити, оскільки Iceberg прикрасив її червоною позначкою з кількістю змін, які чекають на опублікування. У вас їх може бути дві, якщо ви не виконували «*Push*» для метаданих проєкту. Перед вивантаженням Iceberg покаже всі неопубліковані зміни (див. рис. 7.16), а тоді перенесе їх на віддалений

сервер. Він також може перепитати вас ім'я користувача та токен доступу, якщо ви використовуєте HTTPS і виконуєте «*Push*» вперше.

Тепер ви здебільшого закінчили. Проєкт *LightsOut-Game* набув статусу «*Up to date*», а ви знаєте основні аспекти керування своїм кодом за допомогою GitHub, або будь-якого іншого віддаленого сервісу Git. Iceberg розроблено, щоб допомагати вам, тому, будь ласка, прислухайтеся до нього, якщо не знаєте, що робити. Тепер ви готові використовувати послуги, які пропонує GitHub, для покращення якості та контролю коду!

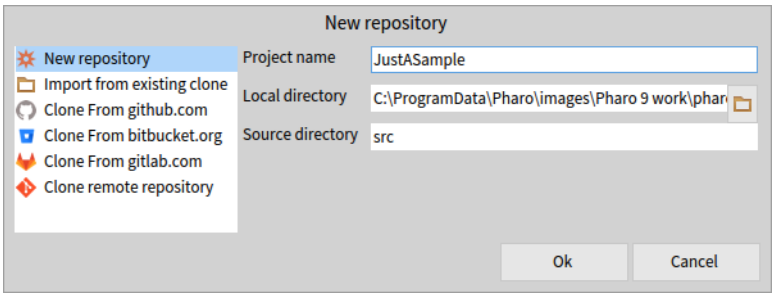

Рис. 7.17. Створення локального сховища у випадку, коли віддаленого ще немає

7.6. Що робити, якщо віддалений репозиторій не створювали?

Ми почали зі створення віддаленого сховища на GitHub, а потім попросили Iceberg додати проєкт, клонувавши його звідти. У результаті отримали віддалене сховище, локальне сховище і робочу копію проєкту. Проте діяти можна і в іншому порядку. Можна створити локальне сховище і зберігати код у ньому. Згодом, коли буде створено сховище на якійсь Git-платформі, код можна буде перенести туди. Створення віддаленого сховища не входить у коло обов'язків Iceberg. Достатньо, що він уміє керувати зберіганням і завантаженням коду.

То як нам досягти такого самого результату, але в оберненому порядку, публікуючи наш локальний проєкт без попереднього віддаленого сховища. Насправді це досить просто, і ми вже так робили в розділі 5, тому давайте пригадаємо.

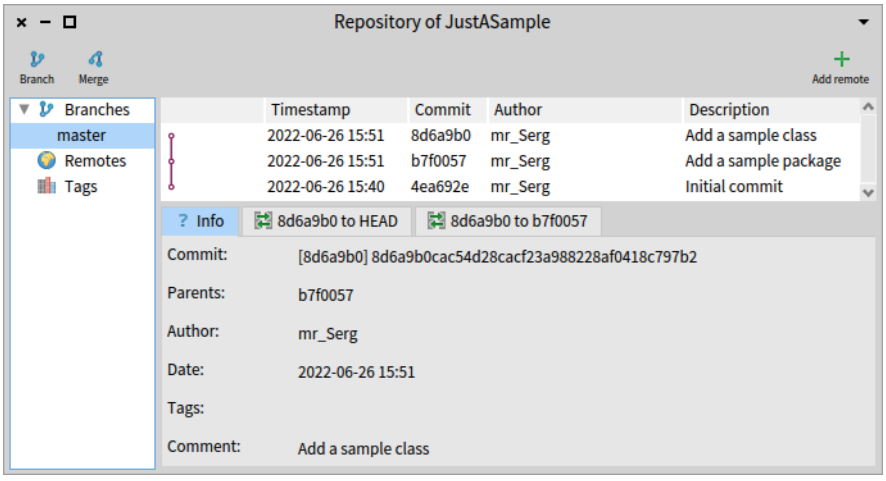

Рис. 7.18. Оглядач репозиторію дає змогу додавати гілки та віддалені сховища, переміщатися між ними

Створіть новий репозиторій

Щоб почати з локального репозиторію, використайте опцію ***New repository*** команди «*Add*» Iceberg, як показано на рис. 7.17 (або 5.9). Ми створимо деякий умовний репози-

торій заради демонстрації. Ви, навіть, можете додати до нього якийсь пакет класів, як ми це вже робили раніше. Після створення збережіть зроблені зміни командою «*Commit*».

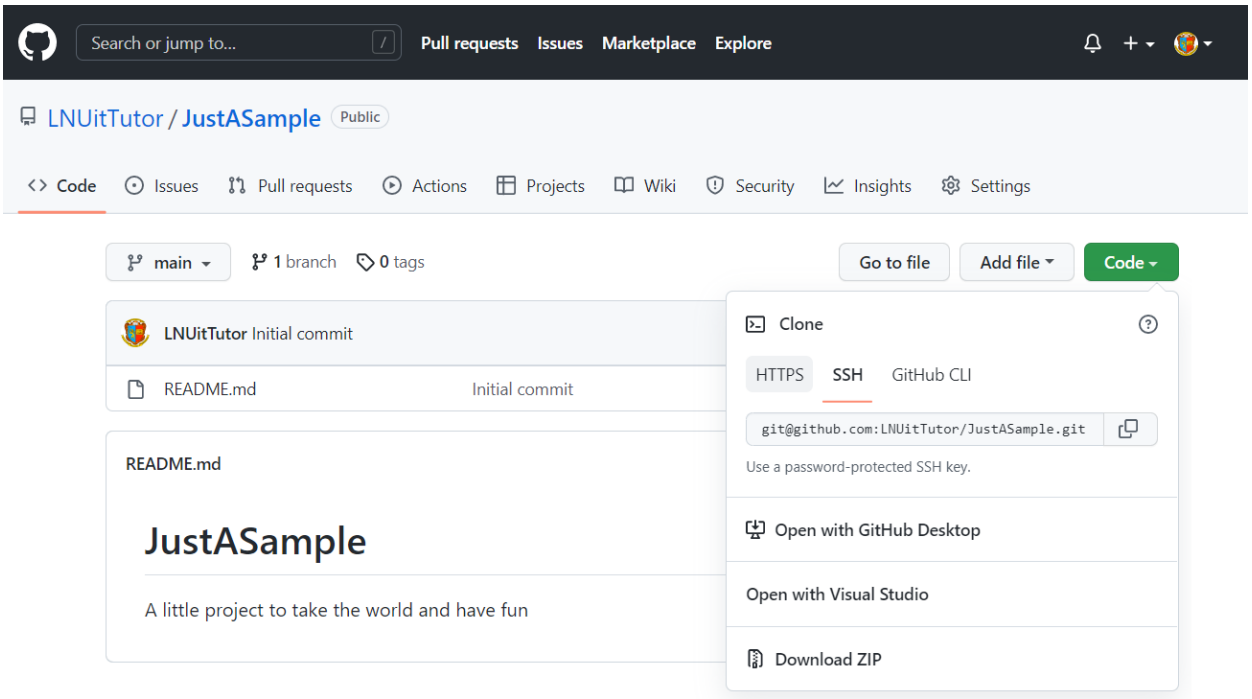

Рис. 7.19. SSH адреса віддаленого сховища на GitHub

Додайте віддалений репозиторій

Якщо ви хочете вивантажувати зміни до віддаленого сховища, то вам доведеться додати його за допомогою оглядача репозиторію. Відкрити його можете командою «*Open Repository*» контекстного меню проєкту або кнопкою ***Repository*** оглядача робочої копії. Оглядач сховища надає вам доступ до пов'язаних з вашим проєктом сховищ Git: ви можете отримати доступ до гілок, керувати ними, а також додавати або вилучати віддалені репозиторії. На рис. 7.18 показано оглядач сховища нашого проєкту.

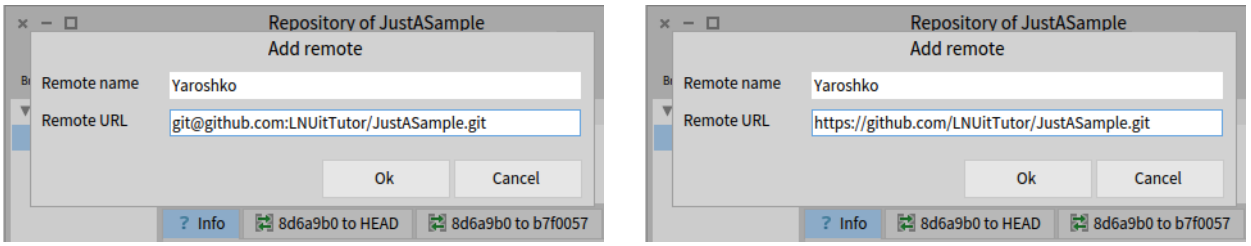

Рис. 7.20. Додавання віддаленого репозиторію в оглядачі сховища
(SSH варіант ліворуч та HTTPS – праворуч)

Віддалене сховище додають кнопкою ***Add remote***. Вона відкриває невеликий діалог, щоб ви могли ввести його координати. Ми починали зі створення локального сховища, тому саме час зайти на GitHub і створити новий порожній репозиторій та довідатися його координати (див. рис. 7.19, або 5.17). Справді, не додавайте до сховища ніяких файлів, навіть *README*, адже ми домовилися починати з Pharo. Натисніть ***Add remote*** і вкажіть координати сховища, як на рис. 7.20.

Вивантажте проєкт у віддалений репозиторій

Тепер ви можете вивантажити усі збережені в локальному сховищі зміни до віддаленого – просто натисніть кнопку ***Push***. Одразу після вивантаження ви побачите в оглядачі сховища, що у вашого проєкту з'явився віддалений репозиторій, як на рис. 7.21.

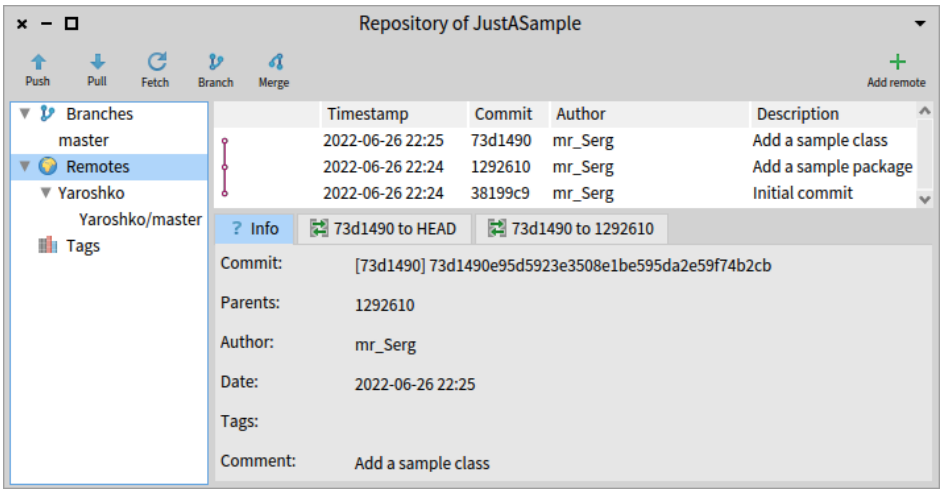

Рис. 7.21 Ви вивантажили проєкт у віддалене сховище

Від перекладача. Одне з правил GitHub – додавати до кожного репозиторію файл *README*. Можливо, ви відгукнетеся на наполегливі пропозиції GitHub і створите такий файл. Тоді віддалене сховище отримає оновлення, а локальне – ні. Щоб їх синхронізувати, потрібно виконати кілька простих кроків: натисніть кнопку *Pull* в оглядачі робочої копії (вона розташована праворуч від *Push*), у вікні діалогу завантаження змін натисніть кнопку *Fetch*, щоб отримати перелік змін з віддаленого сервера, і натисніть кнопку *Pull* діалогу. Тепер сховища знову синхронізовані!

7.7. Конфігурування вашого проєкту

Контроль версій коду – це лише перша частина роботи, щоб ви та інші розробники могли завантажити та використати ваш код. Тепер опишемо, як визначити *Baseline*: карту проєкту, яку ви будете використовувати для визначення залежностей у вашому проєкті та його залежностей від інших проєктів.

Визначення *BaselineOf*

Baseline описує архітектуру проєкту. Виразити залежності між вашими пакетами та іншими проєктами потрібно для того, щоб користувач міг завантажити ваш проєкт без вивчення залежних проєктів чи зв'язків між ними.

Карта проєкту має бути підкласом *BaselineOf*, розміщеним в пакеті «*BaselineOfXXX*», де «*XXX*» – назва вашого проєкту. У попередньому параграфі ми створили простий проєкт *JustASamle* без залежностей. Його карту можна виразити зовсім просто:

```
BaselineOf subclass: #BaselineOfJustASample
   instanceVariableNames: ''
   classVariableNames: ''
   package: 'BaselineOfJustASample'
```

```
BaselineOfJustASample >> baseline: spec
   <baseline>
   spec
      for: #common
      do: [ spec package: 'MySample' ]
```

Як тільки ви визначили карту проєкту, потрібно додати її до проєкту за допомогою оглядача робочої копії, як вже було описано. У підсумку ви мали б отримати щось схоже на зображене на рис. 7.22.

Далі збережіть зміни до локального сховища та вивантажте їх до віддаленого.

Більше інформації про можливості використання *Baseline* можна знайти на Pharo вікі за адресою https://github.com/pharo-open-documentation/pharo-wiki/.

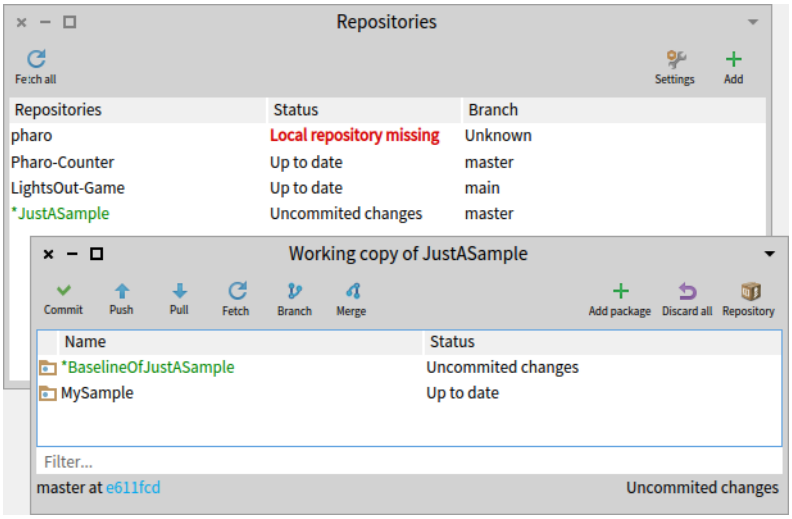

Рис. 7.22. Додавання пакета з картою проєкту за допомогою оглядача робочої копії

7.8. Завантаження з наявного репозиторію

Завантажити опублікований код до нового образу Pharo можна кількома способами.

Завантаження через *Baseline*

Клонуйте в Iceberg віддалене сховище, як ми вже робили в п. 7.4, але створювати метадані не доведеться. Щоб завантажити проєкт в інтерактивному режимі, можете скористатися командою «*Metacello*» контекстного меню (контексто клацніть на назві клонованого сховища в Iceberg, і побачите його). Вона завантажує карту проєкту та виконує її для завантаження пакетів проєкту. Так ви будете впевнені, що всі необхідні підпроєкти також завантажені.

Завантаження вручну

Іноді може виникнути потреба безпосередньо завантажити певний пакет. Як це зробити? Схожою є ситуація з завантаженням проєкту, в якому не визначено *Baseline*. Щоб завантажити конкретний пакет за допомогою Iceberg, потрібно виконати такі кроки:

- додайте проєкт у Iceberg, як ми пояснювали раніше;
- відкрийте оглядач робочої копії подвійним клацанням на імені проєкту;
- позначте потрібний пакет і оберіть команду «*Load*» з його контекстного меню.

Програмне завантаження

Ще один спосіб завантаження полягає в програмному надсиланні повідомлень, як у фрагменті нижче, екземплярові класу *Metacello*:

```
Metacello new
   baseline: 'JustASample';
   repository: 'https://github.com/LNUitTutor/JustASample/src';
   load
```

Це все, що потрібно зробити, щоб завантажити проєкт з визначеною картою (як той, що ми створили в цьому параграфі). Зверніть увагу на те, що в шляху до сховища ми вказали папку, де зберігається код – «*src*» у нашому випадку.

7.9. Оглядаючись назад…

Ви мусите пам’ятати, що коли працюєте у Pharo та змінюєте вміст пакета, керованого Iceberg, то насправді модифікуєте об’єкти, що представляють класи і методи в образі системи, а не змінюєте безпосередньо робочу копію проєкту. Це виглядає так, ніби у вас є дві робочі копії: образ Pharo, завантажений у віртуальну машину, і робоча копія Git на диску.

Коли використовуєте Git для керування своїм проєктом за межами Pharo та Iceberg, наприклад, за допомогою інструмента командного рядка *git*, ви мусите пам’ятати, що код є в образі Pharo *та* в робочій копії (і ще є код у вашому локальному клоні сховища). Щоб оновити образ системи, *спочатку* потрібно оновити робочу копію Git, а *потім* завантажити код з робочої копії в образ. Щоб зберегти свій код, вам *спочатку* треба зберегти код у образі як файли, *потім* додати їх до робочої копії Git, а *потім* зберегти їх у своєму клоні.

Привабливість Iceberg полягає в тому, що він прозоро керує всім цим за вас. Уся виснажлива синхронізація між двома робочими копіями виконується за лаштунками.

Архітектура Iceberg виглядає так:

- ваш код зберігається в образі Pharo;
- Pharo працює як робоча копія (він містить наповнення локального сховища Git);
- Iceberg керує зберіганням вашого коду до робочої копії Git та локального сховища;
- Iceberg керує вивантаженням вашого коду до віддалених сховищ;
- Iceberg керує повторною синхронізацією вашого образу Pharo з локальним сховищем Git, віддаленим сховищем і робочою копією Git.

7.10. Підсумок розділу

У цьому розділі подано найважливіші аспекти того, як правильно розмістити свій код у пакетах та опублікувати його. Це допоможе вам завантажувати його в інші образи та співпрацювати з іншими розробниками Pharo.