個人レベルで GitHub を使う

Seiichi Nukayama

2021-07-09

目次

1 Git for Windows

1.1 インストール

ここ → https://gitforwindows.org/ からダウンロードする。
Git-2.32.0.2-64-bit.exe が **ダウンロード**フォルダにダウンロードされる。[*1]
Git-2.32.0.2-64-bit.exe をダブルクリックすることでインストールが始まる。

2 Eclipse で Github を使う (ワークスペース編)

2.1 ワークスペースを Git で管理する

2.1.1 ワークスペース名の決定

C:ドライブの直下にワークスペースを作成し、それを Git で管理することにする。
ワークスペース名: workspace_git

2.1.2 Github でリポジトリを作成する

Github にサインインして、新しいリポジトリを作成する。
NEW をクリックして始める。
Create a new repository の画面が開く。

Repository name
まず、リポジトリ名を入力する。今回は workspace_git とした。
Description には、このリポジトリの簡単な説明を入れておく。今回は、「git 用ワークスペース」としておく。

Public / Private
このリポジトリを公開するかどうかを設定する。今回は Private とする。

Add a README fle
README.md というファイルを作成するかどうかだが、作成しておいたほうが何かと便利。

Add .gitignore
.gitignore というファイルを作成するかどうかだが、これも作成しておいたほうが便利。
これにチェックを入れると、入力欄が表示される。ここでは Java を選択しておく。

Choose a license
どういうライセンスにするかを選択できる。今回は Private すなわち非公開なので、選択しなくてもかまわない。
ちなみに一番ゆるいライセンスは、MIT ライセンス。

[*1] 32bit 版が必要な場合は、 ここ から Git-2.32.0.2-32-bit.exe をダウンロードできる。

Create repository ボタンをクリックすると、リポジトリが作成される。

新しくできたリポジトリの画面が表示される。Code ボタンをクリックする。

https://github.com/xxxxxx/workspace_git.git が表示されるので、コピーしておく。(コピーボタンがあるので、それをクリック)

2.1.3 クローンする

今回は、C:ドライブ直下に Git 用ワークスペースを作成したいので、C:ドライブで コマンドプロンプトを開く。

C:> と表示されるので、git clone と入力してから、Ctrl-v として、先ほどコピーしておいたものを貼り付ける。

```
C:> git clone https://github.com/xxxxxx/workspace_git.git
```

これで、C:ドライブに workspace_git というフォルダが作成されている。

2.1.4 Eclipse でそのワークスペースで作業する

Eclipse を起動して、その今作成したワークスペースを指定する。

そして、そこで何らかのプロジェクトを作成し、コードを書く。

今回は、sample という Java プロジェクトを作成し、src に Main.java というクラスを作成して、以下のコードを書いた。

リスト 1　Main.java

```
1  public class Main {
2    public static void main(String[] args) {
3      System.out.println("テスト");
4    }
5  }
```

2.1.5 .gitignore ファイルを修正する

.gitignore というファイルで、どのファイル・フォルダを Git で同期するかを指定できる。今回は以下のように設定した。

リスト 2　.gitignore

```
1  # Compiled class file
2  *.class
3
4  # Log file
5  *.log
6
7  # BlueJ files
8  *.ctxt
9
10 # Mobile Tools for Java (J2ME)
11 .mtj.tmp/
12
```

```
13 # Package Files #
14 *.jar
15 *.war
16 *.nar
17 *.ear
18 *.zip
19 *.tar.gz
20 *.rar
21
22 # virtual machine crash logs, see http://www.java.com/en/download/help/error_hotspot.
      xml
23 hs_err_pid*
24 /.metadata/
25
26 # add by Seiichi ---------------------- <1>
27 /Servers/
28 .settings/
29 build/
30 bin/
31 .classpath
32 .project
33 META-INF/
34
35 # necessary
36 !jstl-api-1.2.jar
37 !jstl-impl-1.2.jar
```

<1> までの記述は、GitHub が自動で書いてくれたものである。
今回書いたのは、<1> 以下である。ここに書いたファイルやフォルダは同期の対象から外すことができる。
また、先頭に ! のついているものは、同期を外すの反対なので、同期させるという意味である。

2.1.6 ワークスペースの変更を Github に反映させる

C:\workspace_git でコマンドプロンプトを開く。

```
> git status
```

とすると、どのファイル・フォルダが新しく追加されたか、あるいは変更されたかを表示してくれる。

```
C:\workspace_git>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        sample/

no changes added to commit (use "git add" and/or "git commit -a")
```

また、-u オプションをつけると、フォルダの中のファイルも表示してくれる。

```
> git status -u
```

以下のようになる。

```
C:\workspace_git>git status -u
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        sample/src/Main.java

no changes added to commit (use "git add" and/or "git commit -a")
```

*2

*2 もし、.gitignore を修正しなかったら、以下のようになる。

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)

また、このとき、1 行目の On branch main と 2 行目の Your branch is up to date with 'origin/main'. にも注意を払っておく。

これは、「現在、”main” というブランチ (枝) で作業していますよ。」

「あなたのブランチは、 ”origin” の ”main” ですよ。」 という意味である。

”origin” というのは、今回の場合は、

```
https://github.com/xxxxxx/workspace_git.git
```

のことで (この URL の別名)、”main” というのは、このソース群に github がつけた**ブランチ名** である。[*3][*4]

2.1.7 変更ファイルを指定する

変更・追加したファイルをローカル・リポジトリ (このプロジェクトの保管場所。今作業しているフォルダの中に作成されている) に追加する。これを**ステージング**という。

```
> git add . (ピリオド)
```

この . (ピリオド) は、ここにあるものすべて、という意味であるが、変更・追加・削除のあったものを指定したことになる。変更のなかったものは含まれない。

2.1.8 変更を確定し、説明文をつける

次に、この変更を確定する。このときに必ず説明文をつけなくてはならない。これは当然で、この変更追加は何のためなのか、どういう変更なのかを説明しなければ、他の人に伝わらないし、また、自分があとで見ても、説明がなければ理解できない。

```
> git commit -m "Main.java を新しく追加"
```

(use ”git restore <file>...” to discard changes in working directory)
modified: .gitignore

Untracked files:
(use ”git add <file>...” to include in what will be committed)
sample/.classpath
sample/.project
sample/.settings/org.eclipse.jdt.core.prefs
sample/src/sample/Main.java

no changes added to commit (use ”git add” and/or ”git commit -a”)

この場合、Eclipse でのプロジェクトの設定情報まで同期することになる。そうなると、いろいろややこしいことになるので、ここでは src フォルダなどのソース・フォルダのみ同期することにしている。

[*3] ブランチは開発者が、たとえば textsfver2 などとつけることができる。たとえば、main での開発はできた。今度はこの機能を追加したバージョンを作成してみよう。今度の新バージョン開発ブランチを ver2 としよう。そして、main ブランチから ver2 ブランチに切り換えて開発を行うことにしよう。そうすることで、main ブランチの保守作業もできるし、ver2 での開発も同時にできるというわけである。

[*4] main というのは最近になって GitHub が使うようになったデフォルト・ブランチ名である。それまでは、master がデフォルト・ブランチ名だった。

```
workspace_git¿ git commit -m “Main.java を新しく追加” [main fda8839] Main.java を新しく追加 3 files changed, 27 insertions(+) create mode 100644 sample/.gitignore create mode 100644 sample/src/Main.java
```

2.1.9 ネット上の GitHub リポジトリと同期する

これでローカル・リポジトリは新しくなった。しかし、インターネット上に作成した リポジトリ (リモート・リポジトリという) は古いままである。今からリモート・リポジトリを更新する。

以下のコマンドで更新できる。

```
> git push -u origin main
```

ここで -u オプションをつけているのは、のちに出てくる git fetch コマンドで origin の指定を省略できるようにするためである。

以下のように出力されると成功である。

```
workspace_git> git push -u origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 776 bytes | 776.00 KiB/s, done.
Total 7 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/SeiichiN/workspace_git.git
1ece74f..fda8839  main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

ここで git log として、うまくプッシュできたかを確認する。

```
workspace_git> git log
commit fda8839f98b074ab6e449912f2014a4c790faf05 (HEAD -> main, origin/main, origin/HEAD)
Author: SeiichiN <billie175@gmail.com>
Date:   Mon Jul 12 16:23:27 2021 +0900

    Main.java を新しく追加
```

2.2 別の PC で作業を再開する

たとえば、自宅のデスクトップ PC で作業をし、それを GitHub にプッシュしておいたとする。その作業の続きを、出先のノートパソコンでおこなうことにする。

2.2.1 Git クローンする

ノートパソコンの適当なフォルダでコマンド・プロンプトを開き、そこで git clone とする。ここでは、デスクトップでやったときと同じように、C:ドライブのトップにワークスペースを作成することにする。

あらかじめ、ブラウザで GitHub のページを開き、Code ボタンをクリックして、クローンするための URL をコピーしておく。

それから、C:ドライブのトップでコマンド・プロンプトを開き、以下のコマンドを実行する。

```
> git clone https://github.com/xxxxxx/workspace_git
```

2.2.2 Eclipse で新規プロジェクト

Eclipse を起動する。ワークスペースにクローンしておいた workspace_git を指定する。

新規でプロジェクトを作成する。

このとき注意することは、デスクトップ PC で作成したプロジェクト名と同じ名前を指定することである。

すると、クローンしておいたファイルが自動で読み込まれる。今回の例でいうと、src フォルダの中の Main.java が自動的に読み込まれる。

ここで、Main.java を修正したり、他のファイルを追加したりする。

2.2.3 GitHub にプッシュ

以下の作業をして、GitHub にプッシュする。

```
> git status
> git add .
> git commit -m "Main.java を修正"
> git push -u origin main
> git log
```

2.3 もとの PC で作業を再開する

最初の PC で作業を再開することにする。

出先のノートパソコンで変更を加えているので、まず、それをとりこまねばならない。

Eclipse を 開く前に、Git で管理しているワークスペースをコマンドプロンプトで 開く。

```
> cd c:¥workspace_git
C:¥workspace_git>
```

まず、リモート・リポジトリ (github.com/xxxxxx/workspace_git) に変更があるかを問い合わせる。

```
C:¥workspace_git> git fetch
```

変更がある場合、以下のような表示になる。

```
workspace_git> git fetch
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 5 (delta 2), reused 5 (delta 2), pack-reused 0
Unpacking objects: 100% (5/5), 451 bytes | 451.00 KiB/s, done.
From https://github.com/xxxxxx/workspace_git
fda8839..4ee1984  main       -> origin/main
```

ここで git status とする。

```
> git status
```

以下のような表示になる。

```
workspace_git> git status
ブランチ main
このブランチは 'origin/main' に比べて 1 コミット遅れています。fast-forward することができます。
  (use "git pull" to update your local branch)

  nothing to commit, working tree clean
```

ここで、use “git pull” to update your local branch と表示されているのを確認する。
「あなたのローカル・ブランチを更新するには “git pull” しなさい」という意味 (私訳)
そのとおりにする。

```
> git pull
```

以下のような表示になる。

```
workspace_git> git pull
Updating fda8839..4ee1984
Fast-forward
 sample/src/Main.java | 4 ++--
  1 file changed, 2 insertions(+), 2 deletions(-)
```

これで、このフォルダは更新されている。また、git log とすることで、メッセージを確認できる。

```
workspace_git> git log
commit 4ee1984640c5c710d025b25eff46a1c15ede52ec (HEAD -> main, origin/main, ori
gin/HEAD)
Author\hline SeiichiN <billie175@gmail.com>
Date\hline   Mon Jul 12 19\hline10\hline06 2021 +0900

    Main.java を修正

commit fda8839f98b074ab6e449912f2014a4c790faf05
Author: SeiichiN <billie175@gmail.com>
Date:   Mon Jul 12 16:23:27 2021 +0900

    Main.java を新しく追加
( ... 略 ...)
```

これで更新できたので、Eclipse を起動し、プロジェクトを開いて、編集をおこなう。

編集がすめば、また、workspace_git のフォルダにて、以下のコマンドを実行してリモート・リポジトリに変更を同期させる。

```
> git status
> git add .
> git commit -m “変更点を簡潔に記述する”
> git push -u origi main
> git log
```

出先のノートパソコンで 作業の続きをおこなうには、前回ですでにワークスペースはクローンできているので、以下のように作業できる。

```
> cd C:¥workspace_git
> git fetch
> git status
> git pull
> git log
```

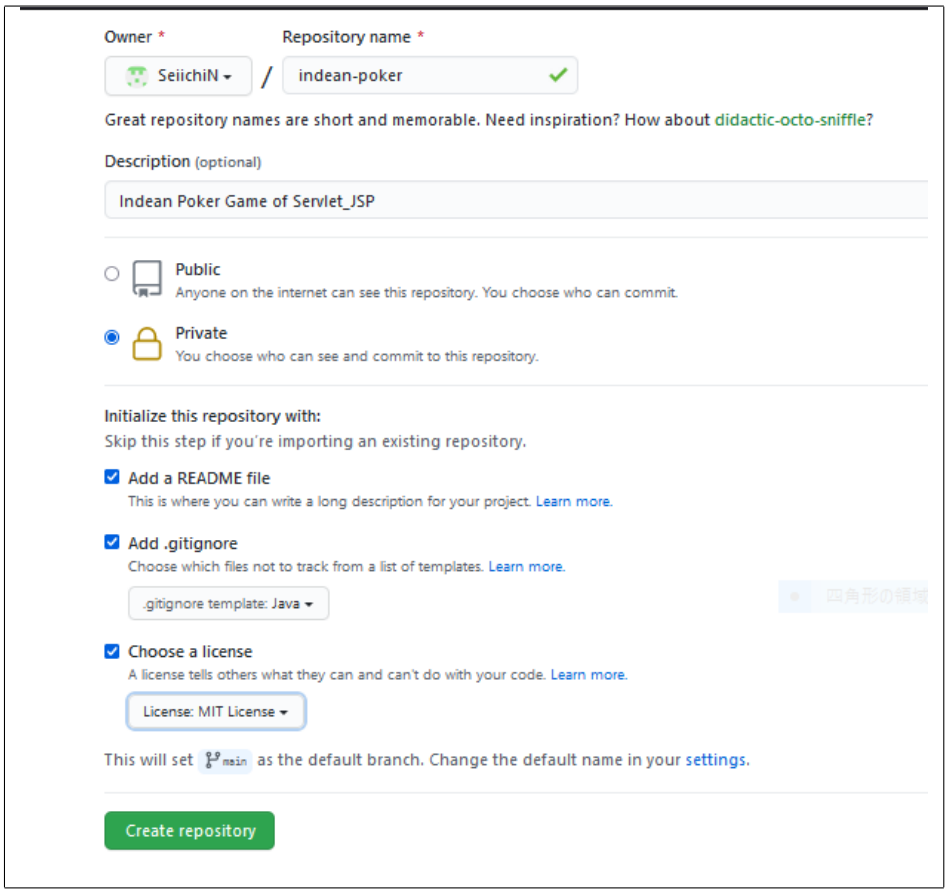
3　Eclipse で Github を使う (プロジェクト編)

3.1　Eclipse のプロジェクトを Git で管理する

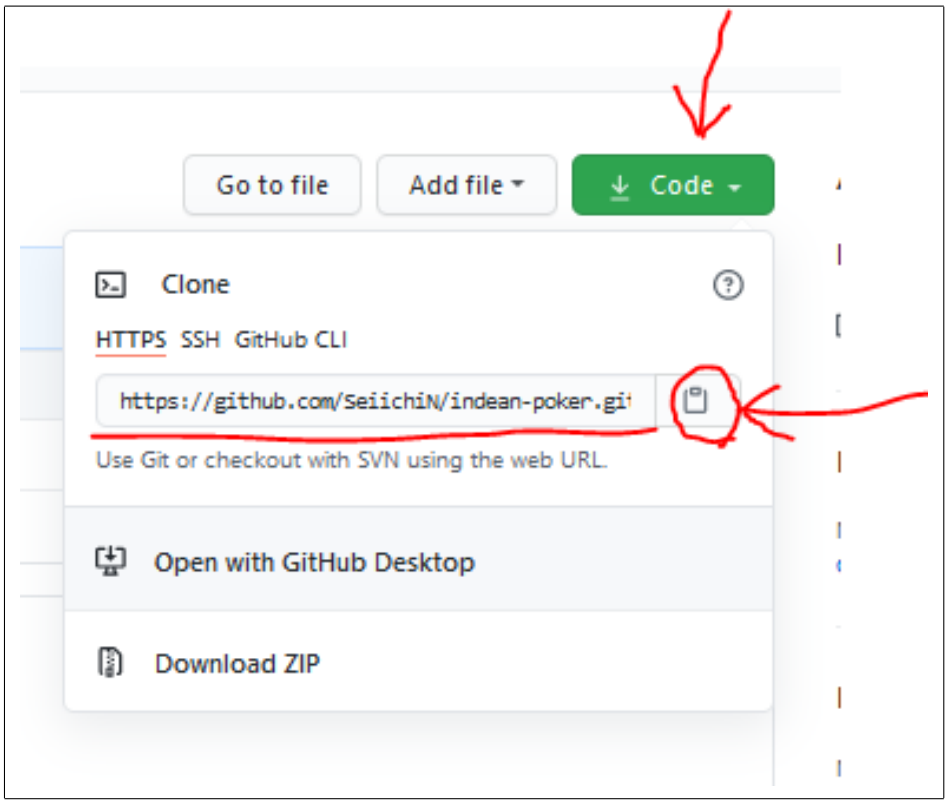
3.1.1　新しい Git リポジトリをつくる

自分の GitHub にアクセスし、新しくリポジトリを作成する。

1. リポジトリ名 — indean-poker
2. Description — Indean Poker Game of Servlet_JSP
3. mode — Private
4. Add a README file — チェック!
5. Add .gitignore — チェック! — Java を選択
6. Choose a lisence — チェック! — MIT Lisence



リポジトリができたら、その URL をコピーする。



3.1.2 Eclipse のワークスペースで git clone

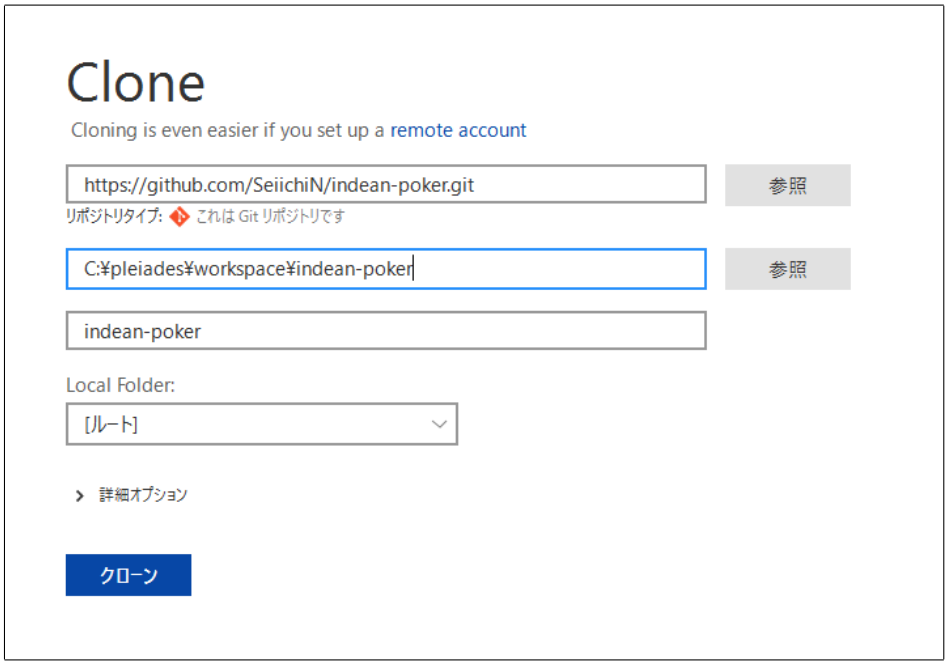
現在使っているワークスペースにクローンすることにする。

sourcetree を起動し、ファイルメニューから「新規/クローンを作成する」を選択する。

クローンを作成するウィンドウが開くので、上から順に以下のように指定する。

1. GitHub でコピーした URL を貼り付ける。
2. Eclipse のワークスペース +indean-poker
   たとえば C:¥pleiades¥workspace ¥indean-poker となる。
3. 名前 — indean-poker
4. Local Folder — ルート

クローン をクリックして実行。



これで、C:¥pleiades¥workspace に indean-poker のフォルダができているはず。

上でやったことをコマンドプロンプトでやるとこうなる。
> cd C:¥preiades¥workspace
C:¥preiades¥workspace> git clone https://github.com/xxxxxxxx/indean-poker.git

3.1.3 Eclipse で新規動的 Web プロジェクトを作成

Eclipse でワークスペースを開き、以下のように動的 Web プロジェクトを作成する。

プロジェクト名	indean-poker
ターゲットランタイム	Tomcat9 あるいは Tomcat8
動的 Web モジュール	4.0 あるいは 3.1
ソースフォルダー	src
Web コンテンツディレクトリ	WebContent

とりあえず、以下のコードを入力する。

リスト 3　/WebContent/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>インディアン・ポーカー</title>
</head>
<body>
        <h1>インディアン・ポーカー</h1>
        <form action="<%=request.getContextPath() %>/play" method="post">
                <input type="hidden" name="mode" value="start">
                <input type="submit" value="札をひく">
```

```
        </form>
</body>
</html>
```

リスト 4　src/model/Card.java

```
package model;

import java.io.Serializable;

public class Card implements Serializable {
        private static final long serialVersionUID = 1L;
        private String suit;
        private int number;

        public Card () {}
        public Card (String suit, int number) {
                this.suit = suit;
                this.number = number;
        }

        public String toString() {
                return suit + ":" + number;
        }

        public String getSuit() {
                return suit;
        }

        public void setSuit(String suit) {
                this.suit = suit;
        }

        public int getNumber() {
                return number;
        }

        public void setNumber(int number) {
                this.number = number;
        }
}
```

リスト 5　src/servlet/Play.java

```
package servlet;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
```

```
13 import javax.servlet.http.HttpServletRequest;
14 import javax.servlet.http.HttpServletResponse;
15
16 import model.Card;
17 import util.Const;
18
19 @WebServlet("/play")
20 public class Play extends HttpServlet {
21         private static final long serialVersionUID = 1L;
22
23         protected void doPost(HttpServletRequest request, HttpServletResponse response
       ) throws ServletException, IOException {
24                 request.setCharacterEncoding("UTF-8");
25                 String mode = request.getParameter("mode");
26                 if (! mode.equals("start")) {
27                         response.sendRedirect("/index.jsp");
28                         return;
29                 }
30
31                 Card card = new Card("club", 12);
32                 request.setAttribute("card", card);
33                 request.getRequestDispatcher("/WEB-INF/jsp/play.jsp").forward(request,
        response);
34
35         }
36
37 }
```

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta charset="UTF-8">
7  <title>インディアン・ポーカー</title>
8  </head>
9  <body>
10         <h1>インディアン・ポーカー</h1>
11         <p>あなたのひいたカード</p>
12         <p>${card.suit}:${card.number}</p>
13 </body>
14 </html>
```

3.1.4 .gitignore ファイルを編集

src フォルダと WebContent フォルダだけを Git で管理したいので、.gitignore ファイルに以下を追加する。

リスト 6 .gitignore

```
1 (... ここまでは省略...)
2
3 # ここから下を追加
4 /.metadata/
5 /Servers/
6 .settings/
```

```
build/
bin/
.classpath
.project
META-INF/

# necessary
!jstl-api-1.2.jar
!jstl-impl-1.2.jar
```

src フォルダは全て Git の管理下においていいが、WebContent の場合は META-INF フォルダは Git の管理から外したほうがいい。

以上のようにすることで、たとえば Tomcat8 環境 (Java1.8) で作成した web アプリのソースコードを Tomcat9 環境 (Java11) のプロジェクトで読み込むことも可能となる。