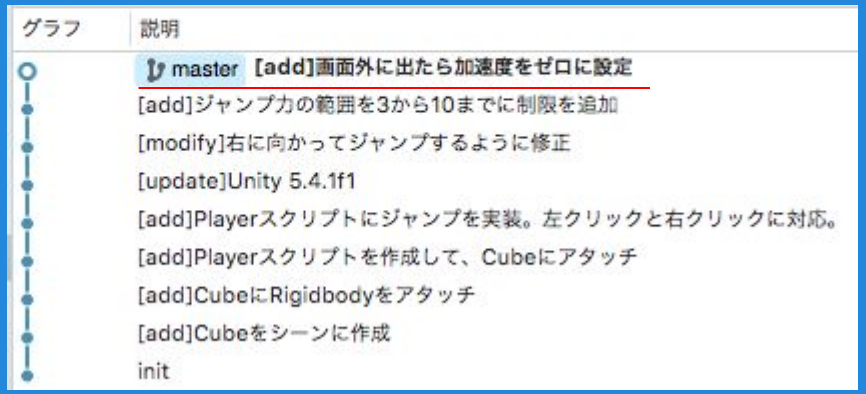
第4章
Gitでバージョン管理
～ブランチ編～

私「新機能～実装♪…まだ適用しないんだった。」
Git「余裕の並行作業、望む時に適用するがよい。」

今回の目的

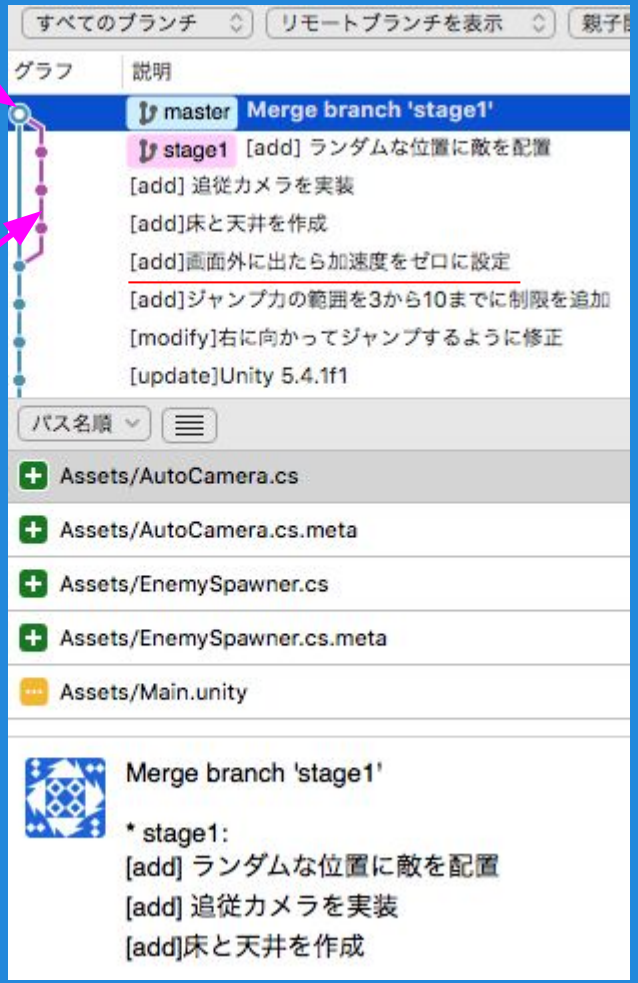
- ブランチの作成（*branch*）
  作業用のブランチを作成して、このブランチに対してコミットします。他のブランチには一切影響を与えないので、自由にコミットができます。

- マージの適用（*merge*）
  現在のブランチに、別ブランチの変更を適用します。
  差分だけが適用された形でコミットされます。



マージ

別ブランチ



実施内容

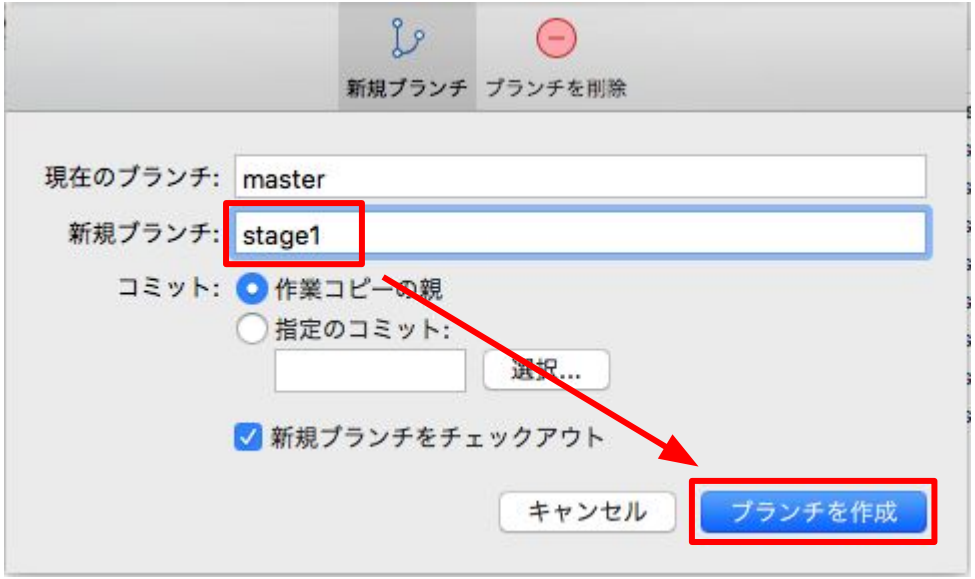
- 別ブランチ(stage1)を作成する！
- 別ブランチで実装とコミットを3回する！
- メインブランチ(master)に
  3回分の変更をマージで適用する！

ブランチの作成（branch）

1. masterブランチを選択して、ブランチボタンをクリックします。

ブランチの作成（*branch*）

2. 新規ブランチに"stage1"と入力して、ブランチを作成します。



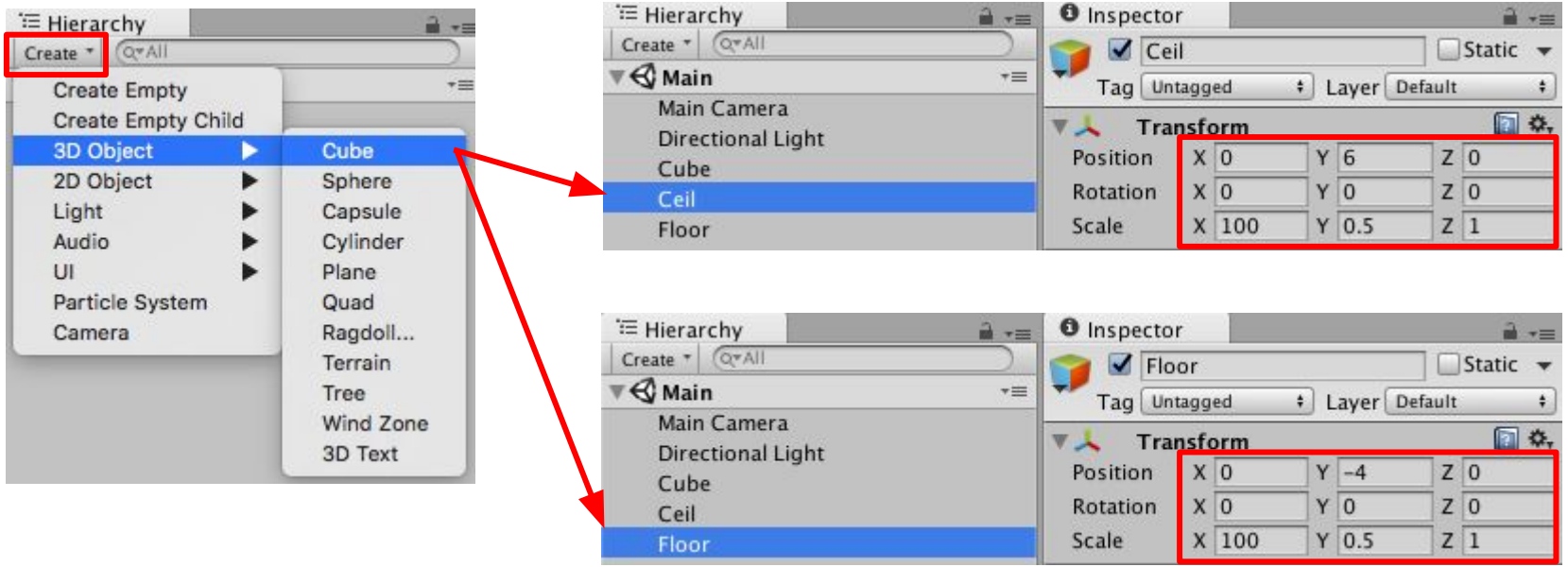
ブランチの作成（branch）

3. stage1ブランチがmasterブランチと同じ位置に作成されたことを確認します。

4. stage1ブランチが現在のブランチになっていることを確認します。

コミット1（床と天井の作成）

1. Cubeを2つ作成して、Floor(床)とCeil(天井)のTransformを設定します。



コミット1（床と天井の作成）

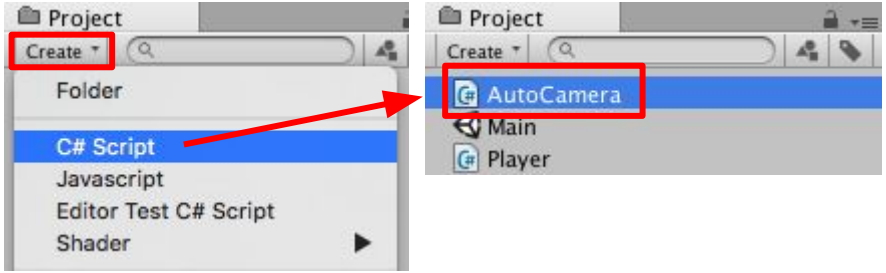
2. **Unityのシーンをセーブして**、stage1にコミットします。

コミット1（床と天井の作成）

3. stage1にコミットされたことを確認します。
（masterより1コミット先行しています。）

コミット2（追従カメラの実装）

1. AutoCamera.csを作成します。



2. AutoCameraを実装します。
（カメラがTarget追従する機能）

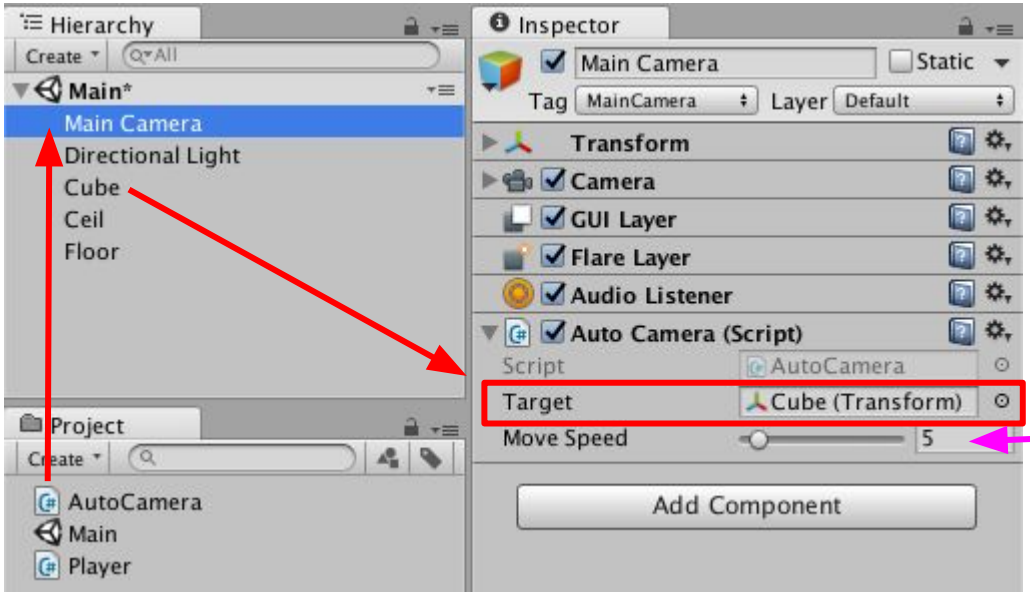
```
using UnityEngine;
using System.Collections;

public class AutoCamera : MonoBehaviour {

    public Transform Target;

    [Range(0f, 60f)]
    public float MoveSpeed = 5f;

    void Update () {
        var pos = transform.position;

        pos.x = Mathf.Lerp(
            transform.position.x,
            Target.position.x,
            Time.smoothDeltaTime * MoveSpeed);

        transform.position = pos;
    }
}
```

カメラの
追従速度

カメラ位置からTarget位置に向けて
x座標のみをスムーズに追従させる

コミット2（追従カメラの実装）

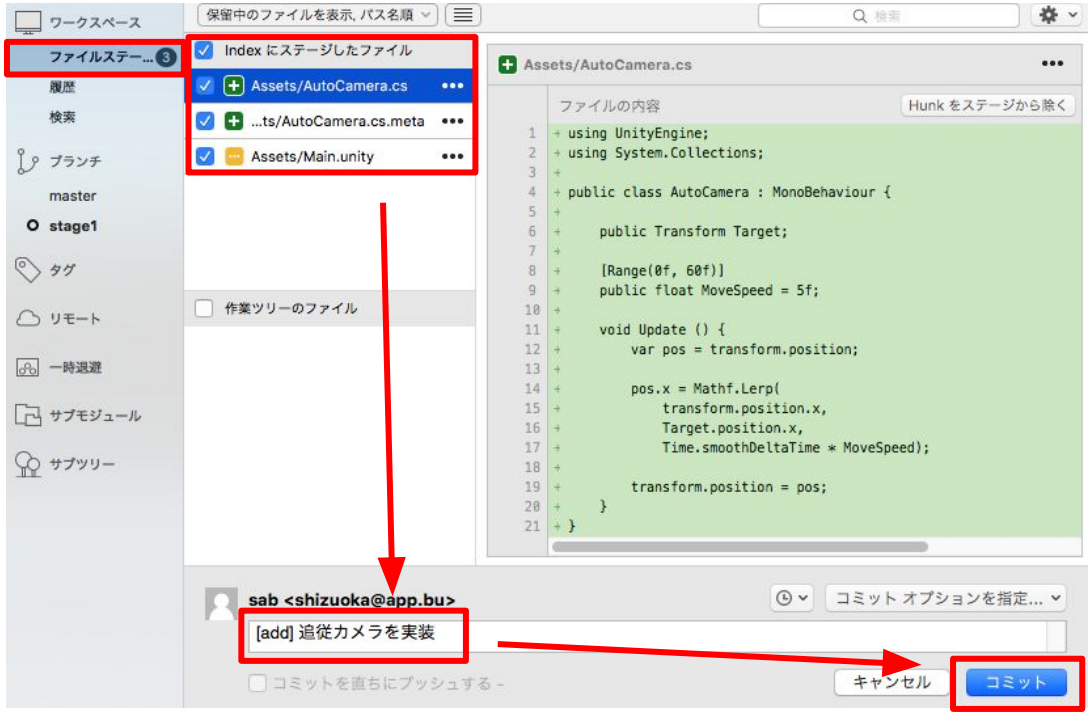
3. AutoCameraをカメラにアタッチします。
4. アタッチしたAutoCameraのTargetにCubeを設定します。



MoveSpeedが
0だと追従しない、
60だと瞬間移動で
追従する

コミット2（追従カメラの実装）

5. **Unityのシーンをセーブして**、stage1にコミットします。



コミット2（追従カメラの実装）

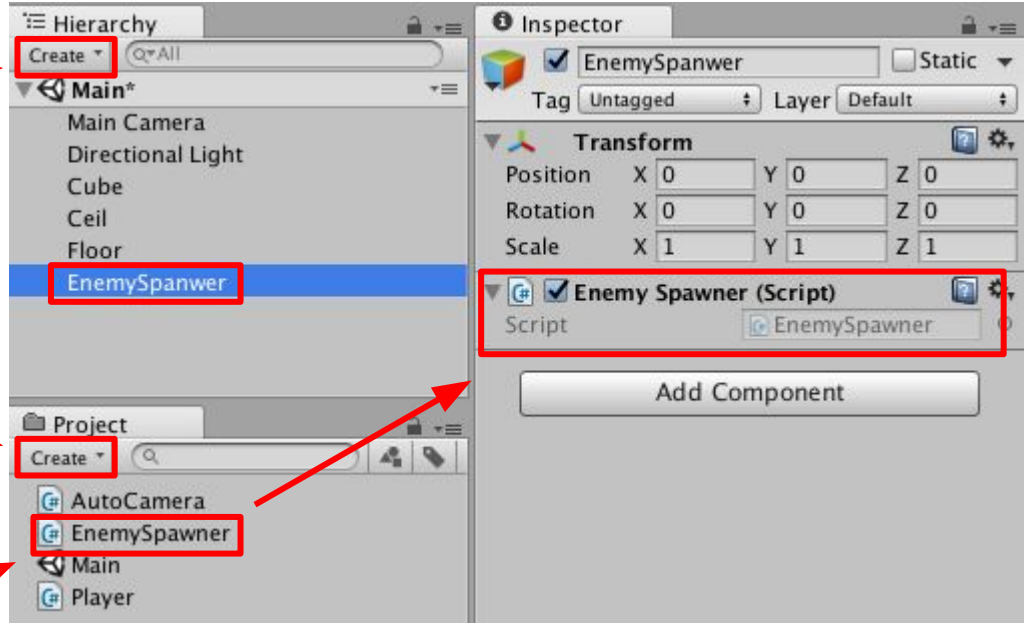
6. stage1にコミットされたことを確認します。
（masterより2コミット先行しています。）

コミット3（敵配置の実装）

1. Create Emptyを選択して
   EnemySpawner
   ゲームオブジェクトを
   作成します。

2. C# Scriptを選択して、
   EnemySpawner.csを
   作成します。

3. EnemySpawnerを
   アタッチします。



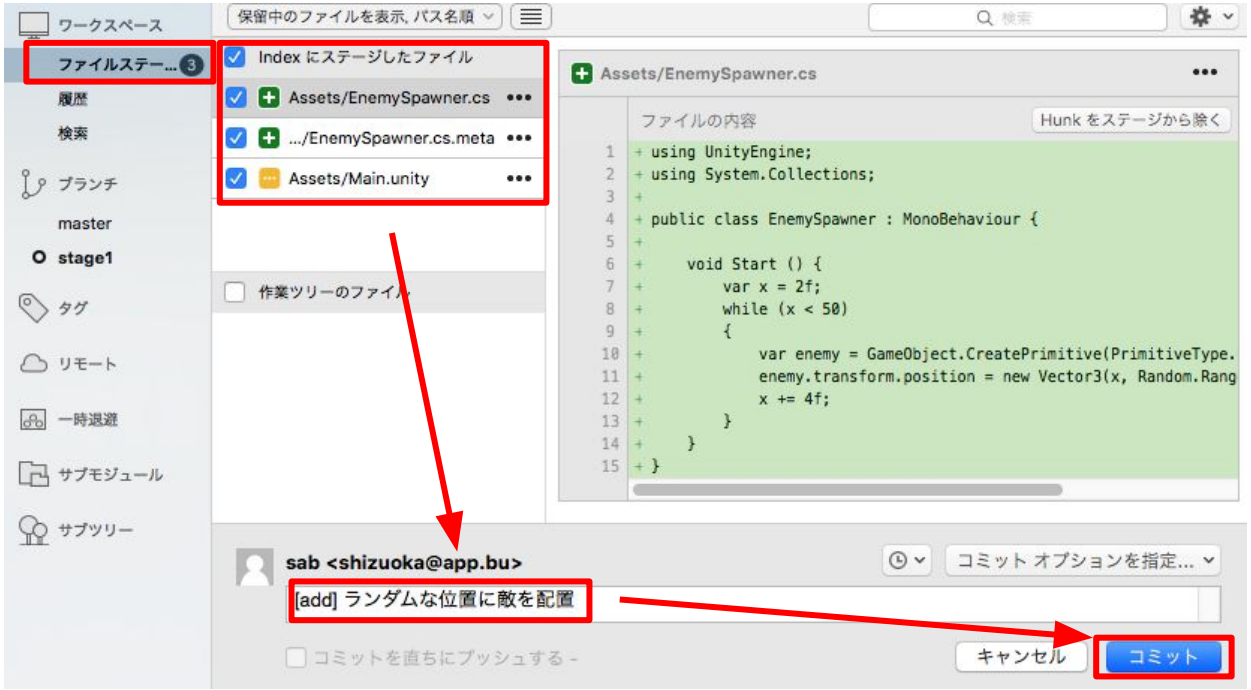
コミット3（敵配置の実装）

4. EnemySpawnerを実装します。
（初期化時に敵(シリンダー形状)をランダムで配置します。）

```
1 using UnityEngine;
2 using System.Collections;
3
4 public class EnemySpawner : MonoBehaviour {
5
6     void Start () {
7         var x = 2f;
8         while (x < 50)
9         {
10            var enemy = GameObject.CreatePrimitive(PrimitiveType.Cylinder);
11            enemy.transform.position = new Vector3(x, Random.Range(-3f, 5f));
12            x += 4f;
13        }
14    }
15 }
```

コミット3（敵配置の実装）

5. **Unityのシーンをセーブして**、stage1にコミットします。



コミット3（敵配置の実装）

6. stage1にコミットされたことを確認します。
（masterより3コミット先行しています。）

マージの適用（*merge*）

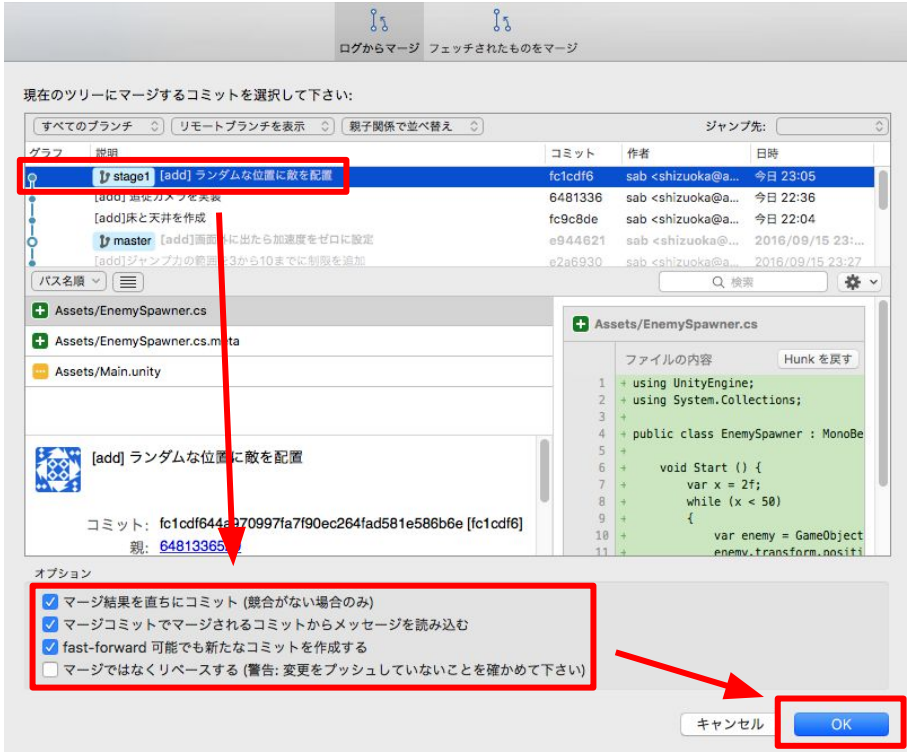
1. masterをダブルクリックして、現在のブランチをmasterにします。
2. マージをクリックします。

マージの適用（*merge*）

3. stage1(最新のコミット)を
   選択します。

4. オプションをチェックします。
   (今回は結果が分かりやすい
   ように設定しています。)

5. OKをクリックします。



ブランチのマージ完了

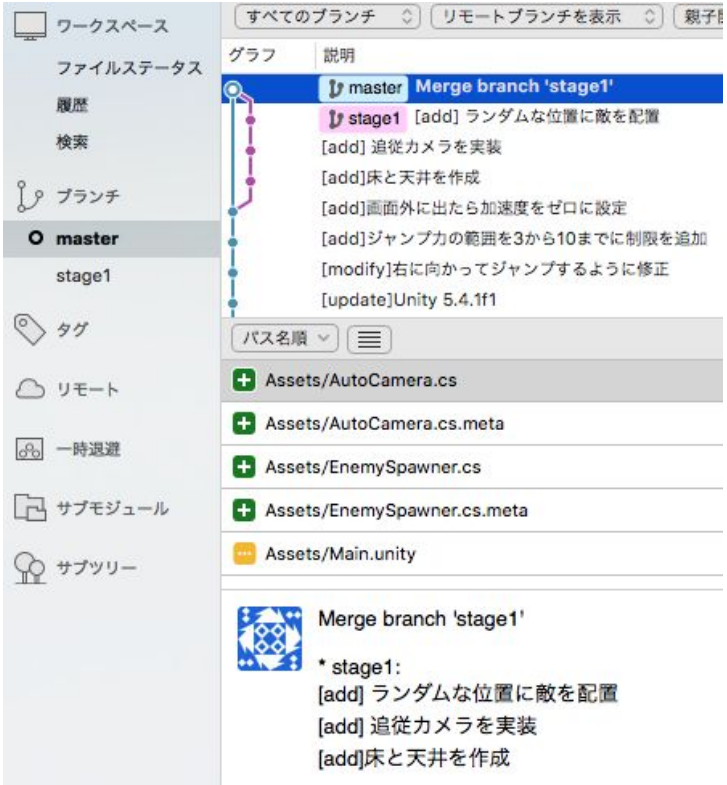
stage1(3回のコミット)の変更が、

masterに反映されました。

masterブランチと分けることで

stage1の実装作業が中途半端でも、
いつでもmasterの状態に戻って
バグ修正することもできます。

複数の別ブランチを作成することも
できます。



補足

- マージ時の競合（コンフリクト）
  別ブランチ(相手)を現在のブランチ(自分)にマージするとき、互いに同じ箇所を変更していると競合が発生します。相手と自分のどちらの変更を反映するか判断できないためです。

- 並行作業のファイルを分担
  複数のブランチで同じファイルを変更しないようにして、競合を防ぎましょう。担当ファイルを分けることが最も安全です。

- 競合が解決しにくいファイル
  ソースコードなどのテキストファイルは、競合が発生した行を手動編集して解決できます。シーンファイルや画像ファイルなどは、相手か自分のどちらかで反映する必要があります。(行単位ではなくファイル単位で解決)

まとめ

- ブランチの作成（*branch*）
  同時にいくつもの並行作業ができます。マージするまでは別ブランチに影響を与えません。

- マージの適用（*merge*）
  別ブランチ(コミット)の変更を反映できます。
  複数のブランチで並行作業したものを、好きなタイミングで必要なものだけ反映することができます。

第4章クリア

最強無敵のGitでブランチを作成して他に影響を与えずに作業ができました。検討作業や修正作業などブランチを分けて、遠慮なく変更していきましょう。

次回は、ログ・リバート・リセットを使って、履歴の確認や打ち消しをしてみましょう。