Git_Api

1.初始化仓库

```
git init      在当前目录创建一个新的Git仓库。
```

2.设置全局邮箱和用户名

```
git config --global user.name “你在GitHub上的用户名”
git config --global user.email “你的邮箱”
```

3.ssh相关操作

3.1.检查是否存在ssh

```
cd ~/.ssh
```

3.2.生成ssh

```
ssh-keygen -t rsa -C “你的邮箱”
```

3.3.测试和仓库的ssh连接

```
ssh -T git@github.com
```

4.与远程仓库的操作

4.1.连接远程仓库

```
git remote add origin “你的仓库SSH地址”
```

4.2.取消连接

```
git remote remove <remote-name>
git remote remove origin
```

4.3.拉取代码

```
git pull origin master
```

4.4.提交相关

```
# 检查暂存区文件
git status

# 添加到暂存区
git add .

# 提交到本地仓库
git commit -m “你的备注信息”

# 提交到远程仓库
git push -u origin master
```

5.分支操作

5.1.修改要提交的分支

```
git checkout <branch-name>
```

5.2.合并分支到主分支（前提条件：你现在是主分支）

```
git merge "你要合并的分支名"
```

Unity_Api

1.Input [输入类]

1.1.获取鼠标按键

```
/*获取鼠标按键三种状态.[0:左键  1:右键  2:中键]*/

[s][bool] GetMouseButtonDown(int)  //按下某键的一瞬间,返回true.

[s][bool] GetMouseButton(int)    //按下某键后,持续返回true.

[s][bool] GetMouseButtonUp(int)   //抬起某键的一瞬间,返回true.
```

1.2.获取键盘按键

```
/*获取键盘按键三种状态.*/
[s][bool] GetKeyDown(KeyCode)        //按下某键的一瞬间,返回true.
[s][bool] GetKey(KeyCode)            //按下某键后,持续返回true.
[s][bool] GetKeyUp(KeyCode)          //抬起某键的一瞬间,返回true.
```

2.MonoBehaviour[Mono行为类]

2.1.一些生命周期函数

```
[void] Start()                          //项目运行之后,自动执行一次.
[void] Update()                         //每帧执行一次,一秒钟大约60次.
[void] FixedUpdate()                    //固定更新,0.02秒执行一次.
```

2.2.碰撞函数

```
[void] OnCollisionEnter(Collision)    //碰撞开始，执行一次.
[void] OnCollisionStay(Collision)     //碰撞进行中，每帧都会执行.
[void] OnCollisionExit(Collision)     //碰撞结束，执行一次.
```

2.3.触发函数

```
[void] OnTriggerEnter(Collder)        //触发开始，执行一次.
[void] OnTriggerStay(Collder)         //触发进行中，每帧都会执行.
[void] OnTriggerExit(Collder)         //触发结束，执行一次.
```

2.4.GameObject对象

```
[GameObject] gameObject   //只读属性,当前脚本组件所挂载到的游戏物体对象.
```

2.5.启用和禁用

```
[bool] 组件对象.enabled    //读写属性,控制组件启用[true]与禁用[false].
```

3.GameObject[游戏物体类]

3.1.[string] name 名字字段

```
[string] name          //读写属性,获取当前游戏物体的名称,修改游戏物体名称.
```

3.2.查找 `GameObject` 对象的函数

```
[s][GameObject] Find(string)    //通过名称查找获取场景内的游戏物体对象.

GameObject.Find(string)
```

3.3.获取当前游戏物体上的某个组件对象

```
[T] GetComponent<T>()           //获取当前游戏物体上的某个组件对象.
```

3.4.控制游戏物体显示

```
[void] SetActive(bool)          //控制游戏物体显示[true]与隐藏[false].
```

4.Debug

```
[s][void] Log(object)                    //在控制台输出信息.
```

5.Transform[变换组件]

5.1.位置

```
[Vector3] position                  //读写属性,获取位置,修改位置.
```

5.2.旋转

```
[Quaternion] rotation               //读写属性,获取旋转数据,修改旋转数据.
```

5.3.获取物体自身坐标系的三个正方向

```
[Vector3] forward, right, up    //读写属性,正前方,右方向,正上方.
```

5.4.旋转和移动的函数

```
[void] Translate(Vector3, Space)     //选择坐标系,按指定的方向移动.
space(wprld/self)代表参考世界/自身坐标系
[void] Rotate(Vector3, float)        //沿固定轴向旋转固定角度.
```

6.Rigidbody[刚体组件]

```
[void] MovePosition(Vector3)        //移动到指定位置[当前位置+方向].
[void] AddForce(Vector3)            //世界坐标系方向添加力[方向*力度].
[void] AddRelativeForce(Vector3) //物体坐标系方向添加力[方向*力度].
```

7.Vector3[三维向量]

```
[float]x, y, z                    //公开字段,可以单个读取,不可以单个修改.
/*获取世界坐标系的六个方向*/
[s][Vector3] forward, right, up    //只读属性,前,右,上.
[s][Vector3] back, left, down      //只读属性,后,左,下.
```

8.Quaternion

```
[Vector3] eulerAngles       //读写属性,将当前四元数转换成向Vector3.
[s][Quaternion] Euler(Vector3)     //将Vector3转换成四元数.
```

9.Collision

```
[GameObject] gameObject            //只读属性,物理碰撞到的游戏物体.
```

10.Collider

```
[GameObject] gameObject            //只读属性,物理触发到的游戏物体.
```