# GIT练习手册

GIT练习手册 1

配置管理系统支持项 2

应用生命周期活动（ALM） 2

配置管理基本活动概要 2

项目生命周期活动概要 2

案例说明 3

任务1. 确定项目信息，启动项目，制定配置管理计划 3

任务2. 确定项目需求基线 4

任务3. 确定系统架构 4

任务4. 确定各模块的设计 4

任务5. 编码及单元测试 5

任务6. 系统集成并测试 5

任务7. 发布 6

任务8. 部署与维护 6

任务9. 结束项目 6

任务10. 执行维护项目 6

GIT 命令手册 8

1. 项目准备，建立项目环境 8

1.1. 设置服务器环境 8

1.2. 查看服务器设置 8

1.3. 创建仓库 8

1.4. 创建分支 8

1.5. 仓库结构 9

1.6. 初始化个人工作区 9

2. 项目日常维护 9

2.1. 添加和提交 9

2.2. 查看历史 9

2.3. 回滚 10

2.4. 恢复 10

2.5. 分支操作创建功能分支 10

2.6. 合并修改/提交任务 11

2.7. 创建基线 11

2.8. 比较两个基线的差异 12

可持续发展的企业面对的是如何发展自己的产品，以管理好产品、及其应用生命周期来创造企业价值。在这样的背景下做好配置管理，就是要支持好产品、应用管理。任何产品的实现都离不开项目，支持好每个项目才能支持好产品、应用管理。

**首先要看到软件开发的全局性，将每个概念映射到配置管理的各个活动中**

**然后对各个配置管理活动的方法进行定义**

## 配置管理系统支持项

项目资产重用

新项目初始化

并行开发的分支策略

变更管理，开发环境集成

发布计划与管理

## 应用生命周期活动（ALM）

**项目组合（平台化，组件化战略）**

产品概念（引用哪些基线）

产品可行性（调研与开发并行）

**开发与验证（变更集支持）**

商品化（亮点功能的上市时间）

优化与定制（紧急任务与增强的有序性）

退市（时间与方式）

## 配置管理基本活动概要

* 标识配置项
* 版本管理
* 过程支持
* 变更控制
* 配置审计
* 配置状态报告
* 构造(Build)管理
* 发布管理
* 部署与维护管理

## 项目生命周期活动概要

* 项目规划
* 配置管理
* 项目启动
* 需求开发
* 需求管理
* 系统设计
* 模块设计
* 编码
* 编译
* 测试
* 发布
* 部署
* 维护与更新

## 案例说明

某IT公司进行xMIS平台的规划，目标是在五年内建成行业领导品牌。策略是每年推出一个版本，在前二年使用一个业界当前成熟的技术框架来追赶竞争对手，并在支持三年后退出市场。第三年推出自己的新架构，后三年就要凭借此新技术成为行业领导者，并在上市后再支持五年左右。

项目规划是v1.0成立“火星”项目，v2.0成立“金星”项目，之后进行“月球”项目对v2.0进行维护与支持；v3.0成立“冥王星”项目，之后成立“海王星”项目对v3.0进行维护与支持。

**角色**：CM Admin -- 维护CM服务器；PM -- 负责协调并监控CM计划的有效执行，同时制订系统架构；DEV – 负责实现系统功能并进行单元测试；Tester – 负责测试系统功能。

**测试机**：192.168.1.108 (CentOS v5.3)

### 确定项目信息，启动项目，制定配置管理计划

* + 项目启动时，需要确定项目基线，是全新项目、升级项目、维护项目还是客户定制项目，不同项目的基线会完全不同。还有很重要的一点是重用哪些第三方组件，这些重要的供方原始代码和库是非常重要的项目资产，要做好入库和基线标识工作。

配置管理计划内容：

项目名，**基线名**，**客户方代表**，项目经理，开发成员，识别与标识配置项，

产品演化规划(roadmap)信息，工具及环境信息，编译/**发布计划**的时间点与操作过程，

**基线(包括产品版本，补丁，build号)命名规则**，分支策略、代码流转过程，**变更控制**，

**部署与维护，**状态报告与配置审计，系统及第三方接口信息，备份方案，培训方案。

**＊练习：PM与CM制定项目的配置管理计划草稿，列出要点**

**GIT任务 –** 初始化项目的基础代码，要求各个小组在…/src/simplewar目录下创建各自的文件夹并包含每位小组成员名字命名的文件，要求自己亲自提交属于自己的文件。

这步练习的现实意义在于，让小组成员有统一的开发环境，基于相同的基线开始工作。  
Admin 命令样例：  
git init gitrepo/demo （仓库策略：一个or多个，测试仓库: gitrepo/demo1）

通过查找来明确几个重要的路径位置:

GIT安装目录 - locate git; 仓库位置 - locate git.conf;

工作区、工作拷贝的位置 - locate \*.git

### 确定项目需求基线

* + 将需求文档加入配置库，如需求规格说明书
  + 使用label(标签)标识需求基线，如EARTH\_BL\_REQ\_1.0
  + 使用一个本地工作区取得此基线，今后任意时刻都能方便地取得当时的需求基线
  + 对变更后的需求应用新的基线，并报告所有变更内容

**＊练习：讨论如何进行需求变更的记录与跟踪；建议与变更管理系统整合**

**GIT任务** – Dev命令样例：

git **clone** git://192.168.1.108/root/demo1.git,   
git **status** /gitrepo/demo, git **add** /gitrepo/demogit/new,   
git **move** /gitrepo/demogit/new /gitrepo/demo, git **delete** /gitrepo/demogit/new/a.c,   
git **diff** /gitrepo/demo/readme.txt, git **commit** -m"**xbsoft@gmail.com initial project**" /gitrepo/demo/new, git **log** /gitrepo/demo, git status -v /gitrepo/demo

**GIT任务：团队工作体验（2-3人一组）**- 产生和利用基线**：**

练习如何利用基线：

这步练习的目的在于理解如何在GIT环境支持基线协作，不管敏捷还是传统的流程，都非常有必要维护一套完整且及时更新的的基线，以便其它成员随时可以更新到相应标签表示的版本

git pull ssh://192.168.1.108/gitrepo/gitrepo/demo  
git tag EARTH\_BL\_REQ\_1.0  
git push EARTH\_BL\_REQ\_1.0

git pull EARTH\_BL\_REQ\_1.0

### 确定系统架构

* + Review系统使用的第三方资源
  + 确定本系统目录结构，子组件
  + 将设计文档加入版本控制
  + 可能需要更新配置管理计划的配置库结构部分

**＊练习：讨论并记录如何管理第三方资源**

**GIT任务：**类似任务2

### 确定各模块的设计

* + 分工完成各模块的详细设计并加入版本控制
  + 形成初步的设计基线，与需求基线同样应用标签并创建视图
  + 此时一般会更新需求基线，执行**闭环变更**，按照变更控制流程，对经过CCB批准的需求变更或新增需求进行更新并形成新版本的需求基线，与任务2同样应用标签并创建视图
  + **变更流程，提交界面、解决界面，权限控制，评估机制**

**＊练习：讨论并记录如何在快速响应的同时控制新需求**

**GIT任务：git的两阶段提交**

先add再commit是git的明显特征，也是其独特设计的主要表现之一。

提交时提供变更管理系统的记录ID, 如git commit -m”WorkItem 888” ssh://192.168.1.108/git

### 编码及单元测试

* + 按迭代方式或按功能块完成编码并集成各开发人员的代码，每天验证编译是否通过 - **持续集成**
  + 建议对每个编译版本应用标签
  + 保持项目的节奏

**＊练习：你觉得自己会去做持续集成吗？为什么？**

**GIT任务：团队工作体验（4-5人一组）：！！！提交前先拉取pull最新代码**

练习如何并行修改代码文件：

配置管理的重要原则之一，就是确保入库代码的质量！把好代码质量第一关是每位程序员应尽的义务。所以我们始终要把最不稳定的代码放在本地，只提交有信心的代码到远程！

请使用类似任务1的命令，由每位小组成员完成一次common.h文件的修改，一次自己名字文件的修改。过程中你会需要解决冲突。最后将合并完成的小组目录打个标签，标签的名字是小组名\_v0.1。这是练习中最重要的一个任务，请思考在此任务完成过程中，如果让你再来一次，你能做的更好的地方，或者别人可以做的更好的地方。

命令样例：git pull …, git merge …, git add ., git commit …, git push …

### 系统集成并测试

* + 周期性地集成整个系统，在合适的环境中准备好正确的版本并由测试人员进行验证
  + 对于每个提交测试的软件版本，必须有标签，并有发布说明，描述此版本的功能集，缺陷情况，标签名等信息

**＊练习：如何才能知道哪个版本可用？**建议将TAGS再分成几个类型，如Released, Built, Tested, Failed。但在这几个类型间变化会增加一点工作量。

### 发布

* + 与测试人员及客户确定发布的方式、媒介、位置
  + 与测试人员确定最终发布前进行几轮回归测试
  + 将项目的最终成果发布给客户
  + 可能在中间阶段给客户发布试用版本或是给既有客户发补丁
  + 注意每个发布的版本号与build号，如windows2003-sp2-5912

**＊练习：请写出一个发布说明－ReleaseNotes，明确规定每次发布的内容及位置**

所有给过测试的东西都应该是只读的，禁止再做任何修改，否则就是新的发布。

### 部署与维护

* + 将通过测试环境验证的软件系统部署到生产环境
  + 针对将来的更新，制定并实现相关的策略，尽量透明、平滑地过渡到新版本
  + 如建立历史版本库、可执行文件库，并用相同的标签统一取用；
  + 在拿到供应商版本(代码，可执行文件，发布说明)时，使用导入脚本更新两个配置库。有时间的话，可以通过编译同一版本的源代码，验证你得到的可执行文件与供应商提供的是否大小一致，功能一致。
  + 建立有效的途径，将系统使用过程中的问题或建议与build号一起反馈给开发者

**＊练习：你的真实项目是怎么部署的？出问题时服务工程师如何响应？有什么值得改进的吗？**

### 结束项目

* + 项目的结束往往伴随收款行为，千万不可马虎。
  + 项目经验与教训的总结，对个人与组织的知识积累非常重要。
  + 配置管理需要支持这种需求，做好归档，锁定等工作。

**＊练习：请回忆并思考在项目结束时的配置管理工作，列出三个要点。**

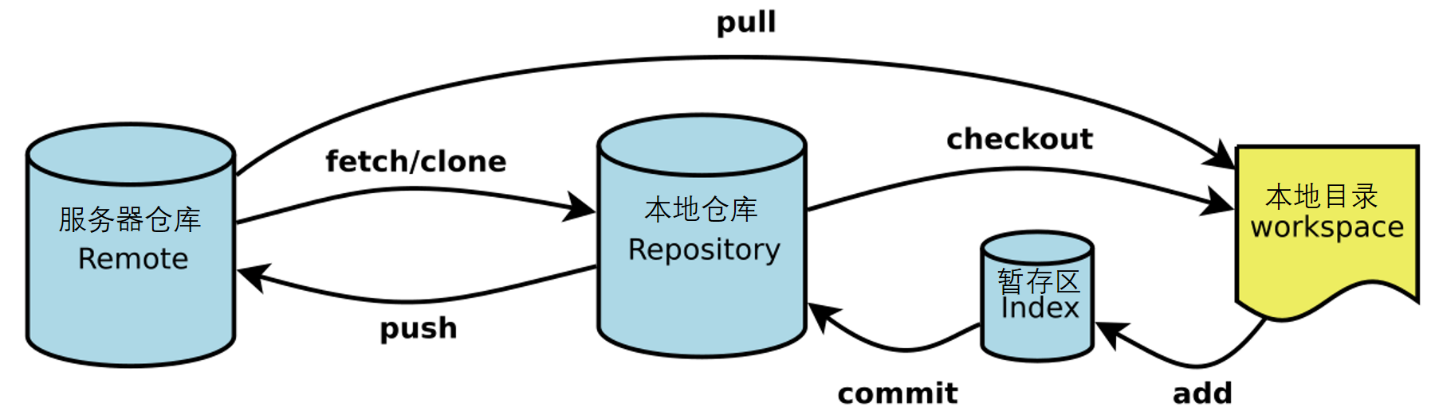
### 执行维护项目

* + 维护项目关乎公司产品质量口碑，需要引起足够重视。
  + 此类项目任务较分散，重点在于分析修改的影响，以决定是否及何时修改。

**＊练习：请列出一个修改请求的流转过程，并说明何时可以关闭，要求说明关闭的依据。**

# GIT 命令手册

本练习模拟一个项目的生命周期，涉及了GIT在Linux环境中对项目提供CM支持的常用操作。完成本练习可以帮助一个普通CM人员顺利上岗。



### 项目准备，建立项目环境

### 设置服务器环境

(可选项) 设置常用命令别名: (.cshrc/.profile).

gl git log  
gc git checkout

将GIT安装目录加入默认路径

setenv PATH=/opt/GIT/bin:$PATH

常用的其它系统命令

du -h, df -k, netstat -apn | grep 8080  
ps -A|grep git, kill xxx

### 查看服务器设置

locate gitconfig

git --version

### 创建仓库

* 项目仓库：

# 本地创建仓库  
git init  
git init /gitrepo/earth

# 创建纯净的仓库（多适⽤于服务器端）  
git init --bare --shared

### 克隆仓库

https://github.com/xbsoft/SimpleMavenJunitWebApp

git clone https://github.com/xbsoft/SimpleMavenJunitWebApp

### 创建分支

如第三方代码分支

git branch 3rdparty  
git [checkout](https://democc/demo/branches/3rdparty) 3rdparty

* 类似添加文件以导入第三方代码

git add .  
git commit -m “add 3rd code”  
git push origin

### 仓库结构

Create folders under project root: src, doc, …

### 初始化个人工作区

# Linux

git config --global user.name “icyleaf”

git config --global user.email “icyleaf.cn@gmail.com”

# Windows的额外配置，异构系统时一定要注意，否则很容易在比较时出现每行都有差异的情况

# 警告提交包含混合换行符的文件，否则容易在异构环境下造成比较时每行都有差异的问题

git config --global core.safecrlf warn

（ or git config --global core.filemode false

git config --global core.autocrlf true ）

### 项目日常维护

### 添加和提交

# touch README  
git add README  
git commit -m “add README”  
touch LICENSE  
git status  
echo “Hello, World” > README  
git add .  
git commit -am “updated README”  
git status

### 查看历史

git log

# 查看最近 3 次提交的详细修改内容  
git log -p -3

# 查看 icyleaf ⽤户最近⼀个星期提交信息  
git log --author icyleaf --since=‘one week ago’

# ⽤简单图形查看分⽀提交的情况  
git log --graph --oneline

### 回滚

# 回滚到最近历史提交的倒数第三个commit  
git log  
git reset --soft HEAD~2  
git log

# 回滚到某个特性的 commit  
git reset --hard {hash}

### 恢复

# echo “Hellp, icyleaf” > README  
git status  
git checkout – README  
git status

# echo “Hello, icyleaf” > README  
git add README  
git reset -- README

# echo “Hello, Mr. icyleaf” > README  
git commit -am ‘Modified README’  
git commit --amend

### 分支操作创建功能分支

# 列出当前所有本地分⽀  
git branch

# 创建功能分支，新建⼀个名为 featureA 的分⽀并切换到它  
git branch featureA  
git checkout featureA

# 下⾯⼀⾏命令等同于上⾯两⾏（快速⾼效，推荐！）  
git checkout -b featureA

# 取到指定基线的内容  
git checkout -b branchname BL1

# 改名分⽀  
git branch -m featureA 2.0/featureA

# 删除分⽀  
git branch -d 2.0/featureA

### 合并修改/提交任务

git checkout -b hotfix

# 开始修复代码

git checkout master

# 合并 hotfix 的代码

**git merge --no-ff hotfix**

# 如果没有冲突⽂件会以 Fast forward 的⽅式顺利合并(--no-ff 的作⽤是保证合并是有多分⽀合并的记录，默认是 --ff，直接把 commit合并到⼀条主线上)

# 如果发⽣冲突，使⽤ git status 查看冲突的⽂件（类似 SVN 状况）

# 解决后，使⽤ git add 标记已完成，并 git commit 提交冲突⽂件

### 创建基线

# 查看当前所有标签  
git tag

# 标记当前分⽀为 v1.0 版本作为归档  
git tag -a v1.0

# 添加带备注的标签  
git tag -a v1.2.1225 -m ‘圣诞节特别版本’

# 把历史特定 commit 标记标签  
git tag -a v1.1 1d2x33

# 查找 v1.0 版本下有多少标记的⼩版本  
git tag -l ‘v1.\*’

# 分享标签  
git push origin v1.0

### 比较两个基线的差异

git diff [v1.0](svn://192.168.1.108/demo/trunk) v1.1

### 基于Git的协作流程

### 推荐团队协作过程

Git的强项体现在团队协作。具体过程与服务器相关，如GitHub或GitLab的Enterprise版本，和BitBucket是常用的Git服务器。

### 协作方式

基于Gitflow流程进行开发团队的协作；

通过分支与环境的对应来进行开发测试的协作；

以业务流程和场景来进行集成与联调等协同，并在测试结果与验收环节，协作完成端到端的业务场景验证。