**git基本操作指南**

[1 服务器端的git 3](#_Toc369532462)

[1.1 安装gitosis 3](#_Toc369532463)

[1.2 添加新仓库 3](#_Toc369532464)

[1.3 添加用户 4](#_Toc369532465)

[2 准备工作 4](#_Toc369532466)

[3 获取与创建项目 5](#_Toc369532467)

[3.1 git clone 获取项目 5](#_Toc369532468)

[3.2 git init 创建项目 5](#_Toc369532469)

[4 基本的快照 5](#_Toc369532470)

[4.1 git add 添加文件到缓存区 5](#_Toc369532471)

[4.2 git status 查看文件在工作目录与缓存的状态 6](#_Toc369532472)

[4.3 git diff 显示已写入缓存与已修改但尚未写入缓存的改动的区别 6](#_Toc369532473)

[4.4 git commit 记录缓存内容的快照 8](#_Toc369532474)

[4.5 git reset 把当前分支指向另一个位置 9](#_Toc369532475)

[4.6 git rm 将文件从缓存区移除 10](#_Toc369532476)

[5 提取与推送 10](#_Toc369532477)

[5.1 git remote 罗列、添加和删除远端仓库别名 10](#_Toc369532478)

[5.2 git fetch 从远端仓库下载新分支与数据 10](#_Toc369532479)

[5.3 git pull 从远端仓库提取数据并尝试合并到当前分支 10](#_Toc369532480)

[5.4 git push 推送新分支与数据到某个远端仓库 10](#_Toc369532481)

[6 分支与合并 11](#_Toc369532482)

[6.1 git branch 列出、创建与管理工作上下文 11](#_Toc369532483)

[6.2 git checkout 切换到新的分支上下文 11](#_Toc369532484)

[6.3 git merge 将分支合并到当前分支 11](#_Toc369532485)

[6.4 git log 显示一个分支中提交的更改记录 12](#_Toc369532486)

[6.5 git tag 给历史记录中的某个重要的点打上标签 12](#_Toc369532487)

[7 检查与比较 13](#_Toc369532488)

[7.1 git log 过滤提交历史记录 13](#_Toc369532489)

[7.2 git diff 查看两个提交快照的绝对改动 13](#_Toc369532490)

[7.3 git show 查看信息 15](#_Toc369532491)

[8 git工作流 15](#_Toc369532492)

# 服务器端的git

git是一个分布式版本控制软件，搭建服务器端的git目的是提供一个git中心仓库。

以下将会介绍如何使用gitosis搭建git服务。

## 安装gitosis

* 1. 假设你已经安装过git，首先通过git获取gitosis：

**$ git clone git://github.com/res0nat0r/gitosis.git**

* 1. 接下来进入gitosis目录，安装gitosis：

**$ cd gitosis**

**$ sudo python setup.py install**

* 1. 添加git用户，用来管理git仓库

**$ useradd git**

* 1. 修改PATH，使git用户可以调用git

**$ vi /home/git/.bashrc**

PATH=/usr/local/bin:/usr/local/git/bin:$PATH

* 1. 创建key pair，并拷贝public key到/tmp下，这样可以确保gitosis-init命令对其有读取权限**[git用户操作]**

**$ ssh-keygen -t rsa**

**$ cp ~/.ssh/id\_rsa.pub /tmp/id\_rsa.pub**

* 1. 以git用户来执行gitosis-init命令**[root用户操作]**

**$ sudo -H -u git gitosis-init < /tmp/id\_rsa.pub**

此时/home/git下增加了两个目录: gitosis、repositories。其中gitosis是gitosis的根目录，repositories是仓库存放目录

* 1. 给脚本post-update赋予可执行权限：

**$ sudo chmod 755 /home/git/repositories/gitosis-admin.git/hooks/post-update**

## 添加新仓库

1. gitosis的管理是通过git来管理的，clone一下

**$ git clone git@localhost:gitosis-admin.git**

进入gitosis-admin目录，有gitosis.conf文件和keydir目录。gitosis.conf是配置文件，用来管理权限控制；keydir目录用来存放用户的public key(.pub文件)

查看一下gitosis.conf，内容如下：

[gitosis]

[group gitosis-admin]

writable = gitosis-admin

members = root@server

其中group代表一个组，writable是仓库名，members是此仓库的成员，可以有多个成员，用空格进行分割

1. 新建一个test仓库

在gitosis.conf文件中添加一下内容：

[group test]

writable = test

members = root@server

把更改提交并push到git@localhost:gitosis-admin.git：

**$ git commit –am "添加新仓库test"**

**$ git push origin master**

在本地创建一个仓库，并push到git@localhost:test.git，gitosis会在/home/git/repositories自动创建test.git这个仓库：

**$ mkdir test**

**$ cd test**

**$ touch README**

**$ git init**

**$ git remote add origin git@localhost:test.git**

**$ git push origin master**

## 添加用户

1. 假设我们要添加的用户为dongnan@dev，需要先创建key

**$ ssh-keygen -t rsa**

假设生成的public key为：~/.ssh/dongnan@dev.pub

1. 修改gitosis.conf，修改后为如下：

[group test]

writable = test

members = root@server dongnan@dev

注意.pub文件名和members中添加的用户名要完全一样

1. 拷贝dongnan@dev.pub到keydir中
2. 把更改push到gitosis-admin.git：

**$ git add \***

**$ git commit -m "添加dongnan@dev到test仓库"**

**$ git push origin master**

1. 接下来dongnan@dev就可以从自己的机器上进行clone了：

**$ git clone git@SERVER\_HOSTNAME:test.git**

# 准备工作

首先，需要确保你已经安装了git，并且已经按照服务器端的git中的方法添加了用户和仓库；

然后，在试用git之前，先配置好自己用户信息：

**$ git config --global user.name "DongNan"**

**$ git config --global user.email "dongnan@aabbcc.com"**

# 获取与创建项目

## git clone 获取项目

获取一个项目使用git clone命令：

**$ git clone git@****SERVER\_HOSTNAME:test.git**

git支持多种协议，如果git中心仓库（服务器端的git）支持，可以使用其他方式clone：

**$ git clone https://SERVER\_HOSTNAME/test.git**

## git init 创建项目

创建一个git项目使用git init命令，需要先创建一个目录：

**$ mkdir test**

**$ cd test**

**$ git init**

此时可以看到项目目录中有个 .git 的子目录。这就是git 仓库，所有有关此项目的快照数据都会存放在这里

# 基本的快照

本地仓库由 git 维护的三棵“树”组成。第一个是你的 工作目录，它持有实际文件；第二个是 缓存区（Index），它像个缓存区域，临时保存你的改动；最后是 HEAD，指向你最近一次提交后的结果。

工作目录

add

缓存区

commit

HEAD

## git add 添加文件到缓存区

在 Git 中，在提交你修改的文件之前，你需要把它们添加到缓存。如果该文件是新创建的，你可以执行 git add 将该文件添加到缓存，但是，即使该文件已经被追踪了 —— 也就是说，曾经提交过了 —— 你仍然需要执行 git add 将新更改的文件添加到缓存去。

以test项目为例，新建文件README，并查看文件状态：

**$ touch README**

**$ git status –s**

?? README

可看到有一个文件处于未被追踪的状态，需要执行git add操作：

**$ git add README**

再执行git status操作查看文件状态，就可以看到README文件已经添加：

**$ git status –s**

A README

如果要添加多个文件，可以使用空格隔开，例如：

**$ git add file\_a file\_b**

如果要添加所有文件，可执行命令：

**$ git add .**

或是：

**$ git add \***

## git status 查看文件在工作目录与缓存的状态

执行 git status 命令可查看文件在缓存与当前工作目录的状态。在git add示例中已经使用，添加 -s 参数的目的是获取较短的输出结果。

如果不加 –s 参数，得到输出结果会是：

# On branch master

#

# Initial commit

#

# Changes to be committed:

# (use "git rm --cached <file>..." to unstage)

#

# new file: README

#

## git diff 显示已写入缓存与已修改但尚未写入缓存的改动的区别

git diff 有两个主要的应用场景。在此介绍其一， 在 **[检查和比较](#_检查与比较)** 一节中将介绍其二。 这里介绍的方式是用此命令描述已临时提交的或者已修改但尚未提交的改动。

1. git diff 查看尚未缓存的改动

**$ vi README**

**$ git status --s**

M README

**$ git diff**

diff --git a/README b/README

index dbee026..1802a74 100644

--- a/README

+++ b/README

@@ -1,2 +1,3 @@

aaa

bbb

+ccc

1. git diff --cached 查看已缓存的改动

git diff --cached 命令查看有哪些内容已经写入缓存了。 也就是说，此命令显示的是接下来要写入快照的内容。所以，如果将上述示例中的 README 写入缓存，因为 git diff 显示的是尚未缓存的改动，所以在此执行它不会显示任何信息。

**$ git status –s**

M README

**$ git add README**

**$ git status -s**

M README

**$ git diff**

**$**

**$ git diff --cached**

diff --git a/README b/README

index dbee026..1802a74 100644

--- a/README

+++ b/README

@@ -1,2 +1,3 @@

aaa

bbb

+ccc

1. git diff HEAD 查看已缓存的与未缓存的所有改动

**$ vi README**

**$ git add README**

**$ vi file\_a**

**$ git status –s**

M README

M file\_a

**$ git diff**

diff --git a/file\_a b/file\_a

index 78c1543..f12e50f 100644

--- a/file\_a

+++ b/file\_a

@@ -1,3 +1,4 @@

aaa

+bbb

111

222

**$ git diff --cached**

diff --git a/README b/README

index 1802a74..51de07c 100644

--- a/README

+++ b/README

@@ -1,3 +1,3 @@

-aaa

+aaa 111

bbb

ccc

使用git diff HEAD 可查看到git diff 和 git diff --cached结果的合集

**$ git diff HEAD**

diff --git a/file\_a b/file\_a

index 78c1543..f12e50f 100644

--- a/file\_a

+++ b/file\_a

@@ -1,3 +1,4 @@

aaa

+bbb

111

222

diff --git a/README b/README

index 1802a74..51de07c 100644

--- a/README

+++ b/README

@@ -1,3 +1,3 @@

-aaa

+aaa 111

bbb

ccc

1. git diff --stat 显示摘要而非整个 diff

**$ git diff --stat**

file\_a | 1 +

1 file changed, 1 insertion(+)

**$ git diff --cached --stat**

README | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

**$ git diff HEAD --stat**

README | 2 +-

file\_a | 1 +

2 files changed, 2 insertions(+), 1 deletion(-)

## git commit 记录缓存内容的快照

使用 git add 命令只是将想要快照的内容写入了缓存，执行 git commit 才是实际存储快照

例如git add示例中：

**$ git status –s**

A README

README文件写入了缓存，但是没有实际存储快照，需要执行git commit 命令：

**$ git commit –m “添加README文件”**

如果你漏掉了 -m 参数，git 会尝试打开一个编辑器以填写提交信息

git commit -a 自动将在提交前将已记录、修改的文件放入缓存区

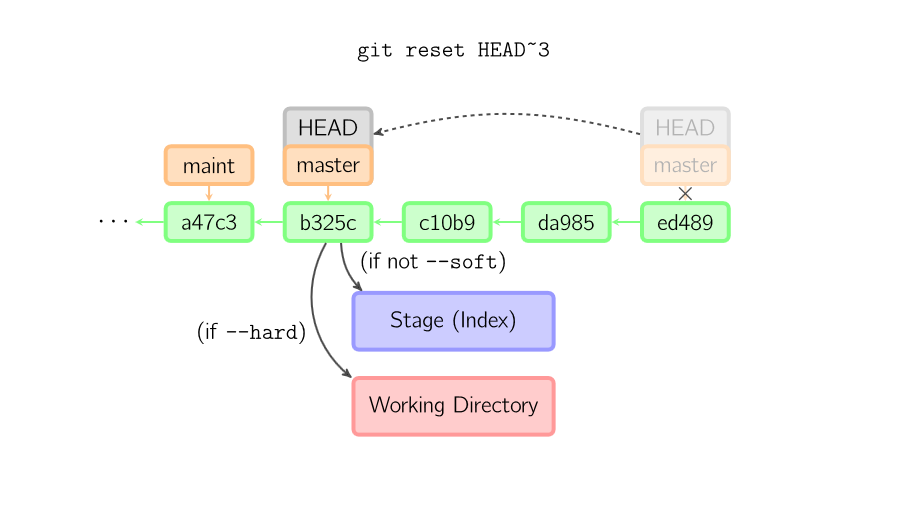
也可以使用快捷命令：

**$ git commit –am “添加README文件”**

## git reset 把当前分支指向另一个位置

reset命令把当前分支指向另一个位置，并且有选择的变动工作目录和索引。也用来在从历史仓库中复制文件到索引，而不动工作目录。

如果不给选项，那么当前分支指向到那个提交。如果用--hard选项，那么工作目录也更新，如果用--soft选项，那么都不变。



1. git reset HEAD 取消缓存已缓存的内容

**$ git status -s**

M README

M file\_a

**$ git reset HEAD**

Unstaged changes after reset:

M README

M file\_a

**$ git status -s**

M README

M file\_a

执行 git reset HEAD 用来取消之前 git add 命令已添加但不希望包含在下一提交快照中的缓存

1. git reset HEAD~3 --hard 将工作目录和索引向前回退3个版本
2. git reset [version] --head 将工作目录和索引会退到指定快照

## git rm 将文件从缓存区移除

git rm 会将条目从缓存区中移除。这与 git reset HEAD 将条目取消缓存是有区别的。 “取消缓存”的意思就是将缓存区恢复为我们做出修改之前的样子。 在另一方面，git rm 则将该文件彻底从缓存区踢出，因此它不再下一个提交快照之内，进而有效地删除它。

总之，执行 git rm 来删除 Git 追踪的文件。它还会删除你的工作目录中的相应文件

默认情况下，git rm file 会将文件从缓存区和你的硬盘中（工作目录）删除。 如果要在工作目录中留着该文件，可以使用 git rm –cached

# 提取与推送

## git remote 罗列、添加和删除远端仓库别名

1. git remote 列出远端别名

**$ git remote**

origin

**$ git remote -v**

origin git@localhost:test.git (fetch)

origin git@localhost:test.git (push)

1. git remote add 为你的项目添加一个新的远端仓库

**$ git remote add origin git@localhost:test.git**

1. git remote rm 删除现存的某个别名

**$ git remote rm origin**

## git fetch 从远端仓库下载新分支与数据

git fetch 会使你与另一仓库同步，提取你本地所没有的数据，为你在同步时的该远端的每一分支提供书签。 这些分支被叫做“远端分支”，除了 Git 不允许你检出（切换到该分支）之外，跟本地分支没区别 —— 你可以将它们合并到当前分支，与其他分支作比较差异，查看那些分支的历史日志，等等。

## git pull 从远端仓库提取数据并尝试合并到当前分支

git pull 命令就是在 git fetch 之后紧接着 git merge 远端分支到当前的活动分支

## git push 推送新分支与数据到某个远端仓库

将改动推送到远端仓库。 执行 git push [alias] [branch]，就会将你的 [branch] 分支推送成为 [alias] 远端上的 [branch] 分支

# 分支与合并

## git branch 列出、创建与管理工作上下文

git branch 命令是 Git 中的通用分支管理工具，可以通过它完成多项任务。 我们先说你会用到的最多的命令 —— 列出分支、创建分支和删除分支。

1. git branch 列出可用的分支
2. git branch -a 列出所有的分支，包括远程分支
3. git branch (branchname) 创建新分支
4. git branch -d (branchname) 删除分支

## git checkout 切换到新的分支上下文

git checkout 命令用来切换到新的分支上下文，工作目录、缓存区、HEAD都将会改变。可用此命令获取将要发布的版本，然后发布版本（上传版本至线上）。

1. git checkout -b (branchname) 创建新分支，并立即切换到它
2. git checkout (branchname) 切换到另一分支

git checkout (branchname) branchname可以是一个远程分支，可通过命令 git branch -a 查看到远程分支

1. git checkout [version] 切换到指定的快照
2. git checkout (tag) 切换到指定的tag

## git merge 将分支合并到当前分支

一旦某分支有了独立内容，终究会希望将它合并到主分支。可以使用 git merge 命令将任何分支合并到当前分支中去。

1. git merge (branchname) -m 将分支branchname合并到当前分支

**$ git branch**

develop

\* master

**$ git checkout develop**

Switched to branch 'develop'

**$ touch file\_f**

**$ git add \***

**$ git commit -m "添加新文件file\_f"**

[develop f511e0d] 添加新文件file\_f

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 file\_f

**$ git checkout master**

Switched to branch 'master'

**$ git merge develop -m "合并develop分支到master：添加新文件file\_f"**

Merge made by the 'recursive' strategy.

file\_f | 0

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 file\_f

1. git merge [version] -m 将快照[version]合并到当前分支

## git log 显示一个分支中提交的更改记录

git 提供了一个获取当前快照的所有提交消息的工具，叫做 git log。

要理解日志（log）命令，需要了解当执行 git commit 以存储一个快照的时候，都有什么信息被保存了。 除了文件详单、提交消息和提交者的信息，git 还保存了此次提交所基于的快照。 也就是，假如克隆了一个项目，是在什么快照的基础上做的修改而得到新保存的快照的？ 这有益于为项目进程提供上下文，使 Git 能够弄明白谁做了什么改动。 如果 git 有该快照所基于的快照的话，它就能自动判断该快照都改变了什么。

1. git log 显示一个分支中提交的更改记录
2. git log --oneline 使用较简洁的方式查看历史记录
3. git log --oneline --graph 开启拓扑图选项查看简洁的历史记录
4. git log --oneline (branchname) 查看指定分支的历史记录
5. git log --oneline (brachA) ^(branchB) 查看当branchA合并到branchB时会有哪些提交会作用到branchB
6. git log --oneline --decorate --graph 查看分支的历史记录,包括添加标签的记录

## git tag 给历史记录中的某个重要的点打上标签

如果达到一个重要的阶段，并希望永远记住那个特别的提交快照，可以使用 git tag 给它打上标签。 该 tag 命令基本上会给该特殊提交打上永久的书签，从而使你在将来能够用它与其他提交比较。 通常，在切取一个发布版本或者交付一些东西的时候打个标签。

-a 选项意为“创建一个带注解的标签”，从而必须为标签添加注解。不用 -a 选项也可以执行的，但它不会记录这标签是啥时候打的，谁打的，也不会让你添加个标签的注解。

1. git tag -a (v1.3) -m 给当前分支的当前快照打上标签，并添加注解

**$ git tag -a v1.3 -m** **'发行版v1.3'**

当执行 git log --decorate 时，就可以看到新添加的标签了

**$ git log --oneline --decorate --graph**

\* f02cae0 (HEAD, tag: v1.3, master) Merge branch 'develop'

|\

| \* 2c8e9cd (develop) 添加文件file\_h

\* | 44c7dbd (tag: v1.2) 合并develop到master:添加文件file\_g

|\ \

| |/

| \* e37a45a 添加文件file\_g

\* | 4dd7a91 Merge branch 'develop'

|\ \

| |/

| \* f511e0d 添加新文件file\_f

\* | 8cf6208 删除文件file\_e

\* | f419020 修改README和file\_a

|/

\* 97f7e4f 添加文件file\_e

…

1. git tag -a v0.9 [version] -m 给某个历史快照打上标签，并添加注解

# 检查与比较

## git log 过滤提交历史记录

1. git log --author 只寻找某个特定作者的提交

要过滤提交历史，只寻找某个特定作者的提交，可以使用 --author 选项。 例如，要找 git 源码中 Linus 提交的部分，可以执行类似 git log --author=Linus 的命令。 这个查找是大小写敏感的，并且也会检索电子邮箱地址。 在此例中使用 -[number] 选项，以限制结果为最近 [number] 次的提交。

**$ git log --author=DongNan --oneline -5**

f02cae0 Merge branch 'develop'

2c8e9cd 添加文件file\_h

44c7dbd 合并develop到master:添加文件file\_g

e37a45a 添加文件file\_g

4dd7a91 Merge branch 'develop'

1. git log --since --before 根据日期过滤提交记录
2. git log --grep 根据提交注释过滤提交记录
3. git log -S 依据所引入的差值过滤
4. git log -p 显示每个提交引入的补丁
5. git log --stat 显示每个提交引入的改动的差值统计

## git diff 查看两个提交快照的绝对改动

要查看两个提交快照的绝对改动，可以用 git diff 命令。这在两个主要情况中广为使用：1、查看两个分支彼此之间的差值；2、查看自发布或者某个旧历史点之后的变化

1. git diff (branchA) (branchB) 查看两个分支间的差值

**$ git diff develop master**

diff --git a/README b/README

index dbee026..1802a74 100644

--- a/README

+++ b/README

@@ -1,2 +1,3 @@

aaa

bbb

+ccc

diff --git a/file\_a b/file\_a

index 7466234..78c1543 100644

--- a/file\_a

+++ b/file\_a

@@ -1,2 +1,3 @@

aaa

111

+222

diff --git a/file\_e b/file\_e

deleted file mode 100644

index e69de29..0000000

1. git diff [version] 查看快照[version]之后的修改

[version] 可以是快照的hash,也可以是自定义的tag标签

**$ git diff 97f7e4f**

diff --git a/README b/README

index dbee026..1802a74 100644

--- a/README

+++ b/README

@@ -1,2 +1,3 @@

aaa

bbb

+ccc

diff --git a/file\_a b/file\_a

index 7466234..78c1543 100644

--- a/file\_a

+++ b/file\_a

@@ -1,2 +1,3 @@

aaa

111

+222

diff --git a/file\_e b/file\_e

deleted file mode 100644

index e69de29..0000000

diff --git a/file\_f b/file\_f

new file mode 100644

index 0000000..e69de29

diff --git a/file\_g b/file\_g

new file mode 100644

index 0000000..e69de29

diff --git a/file\_h b/file\_h

new file mode 100644

index 0000000..e69de29

1. 加上 --stat 参数，显示统计信息

**$ git diff 97f7e4f --stat**

README | 1 +

file\_a | 1 +

file\_e | 0

file\_f | 0

file\_g | 0

file\_h | 0

6 files changed, 2 insertions(+)

## git show 查看信息

1. git show 查看当前提交的版本信息
2. git show [version] 查看指定快照的版本信息
3. git show (tag) 查看指定tag的版本信息
4. git remote show 查看远程仓库信息
5. git remote show [alias] 查看某个远程仓库的信息，[alias]为远程仓库的别名

# git工作流

1. 主要分支

master: 永远处于 production-ready 状态

develop: 最新的下次发布开发状态

1. 辅助性分支

Feature branches: 开发新功能都从 develop 分支出來，完成后 merge 回 develop

Release branches: 准备要 release 的版本，只改 bugs。从 develop 分支出来，完成后merge 回 master 和 develop

Hotfix branches: 等不及 release 版本就必须马上修改 master并上线的情況。会从master 分支出来，完成后 merge 回 master 和 develop

