rge图解Git的基本概念、命令

薛明忠

摘要：第一部分以图文方式介绍git基本对象和操作，展示不同操作模式的生成的版本关系图。第二部分、第三部分是git命令介绍，了解不同git 命令实现的主要功能。

# 基本概念

## Git的数据流及相应的命令：

本地仓库(.git)

工作区

远程仓库repo(.git)

refs/heads/

dev

Bugfix\_xx

…

refs/heads/

dev

…

refs/remotes/repo/

dev

Bugfix\_xx

…

…

push

fetch

checkout

index

stage

commit

工作树

add

stash

版本：

每次提交（commit）都会产生一个新的版本；

git使用SHA1值做为它的唯一编号,可以使用commit号的前4位；

每个版本至少有一个父版本（版本号^1）；

merge产生的版本，将有两个父版本(分别是：版本号^1，版本号^2)；

版本树：依赖于父版本的信息，所有的版本构成一个有向图；没有merge时有向树。

git commit –amend 使用新的版本取代当前HEAD所指向的版本。

标签：是某个版本的引用。

分支：可以说是动态的标签，每次commit时会自动更新到最新的版本上。

仓库：存储git各种对象：用户各版本的信息，各种引用，标签等。

工作区：用户查看、编辑的区域——系统中一个普通的目录树。

暂存区：也就index或stage, 是准备提交到库中的一个临时区域，默认情况下commit命令仅提交本区域中的内容。

stash: 工作区不是“clean”状态时（含有未提交的内容），包括切换分支等命令是不允许执行的。如果不想提交版本到库中，可以将当前工作区的状态保存在这个stash中，保存的内容包括“工作区”和“暂存区”的修改，以便以后可以恢复。

合并代码的方式有这几种：

git rebase 合并代码，将当前分支的拉出位置移动到目标分支的头部，当前分支中所有的节点都合并了目标分支的最新修改。

git merge –ff 在目标分支拉出的位置是当前位置的头部时，将当前分支的位置移动到目标分支的头部。

git merge --no-ff 在当前分支中新建立一个新的提交，这提交饭后 目标分支的修改。

git merge –q 将多个提交的代码合并到当前分支，并保持分支独立。

git cherry-pick 将同样代码变更合入到几个不同的分支，并保持分支的独立。

merge操作默认：如当前分支有自己的修改，使用no-ff的方式，这种情况下使用-ff 会失败，要使用这种方式需要将要合并的分支进行rebase在当前分支的头部后再合入当前分支；当分支无修改，默认使用ff。

git pull 默认使用的是merge，默认策略如上；要想使用git rebase 的试合并，请使用git pull –r， 这个效果和svn中的update类似,提示冲突时，解决冲突后,git add, 然后再git rebase --continue。

## 合并 与 重定位

D0

D1

N1

M1

M2

N2

M

D0

D1

N1

M1

M2

N2

D0

D1

N1

M1

M2

N2

master

my\_br

master

master

my\_br

my\_br

**begin**

**merge master**

**rebase onto master**

左边是初始状态：本地分支从D1, 提交两次M1, M2,服务器上其他人修改了并同步到本地，也修改两次N1, N2。

中间是merge的结果：在my\_br分支下，执行git merge master；merge命令创建了一个新的提交（版本M）

右边是rebase的结果：在my\_br分支下，执行git rebase master; 执行后M1,M2版本中都包含N1,N2的修改。完整命令：git rebase --onto master D1 M2。

pull默认使用merge方式，如果需要使用rebase 请增加选项 -r(或--rebase)。

## 快进与合并

N1

M1

M2

N2

master

my\_br

N1

M1

M2

N2

master

my\_br

N1

M1

M2

N2

master

my\_br

M

**merge my\_br**

merge –no-ff my\_br

begin

最右边是是初始状态。从master分支checkout –b my\_br，修改，并提交了两次。

中间：git checkout master; git merge –no-ff my\_br, 将my\_br修改合并到master分支。

右边：git checkout master; git merge my\_br 这种情况下默认使用ff方式,master分支快进到M2版本。

## Merge 与Merge -q

N1

M1

M2

N2

my\_br

N1

M1

M2

N2

master

my\_br

M

**merge -q my\_br**

merge my\_br

begin

N1

M1

M2

N2

master

my\_br

N3

N3

master

N3

N3

git merge –q my\_br

git commit –m “you new commit msg”

这种方式与普通git merge相比，新的节点上不会有指向my\_br指针，日志上独立。适用情况：本地的my\_br是用于保存中间临时的草稿状态，不需要对外发布的修改日志。

## 相对引用

A

B

C

D

E

F

G

C^0

C^1

C^

C~1

C^2

D^1

D^

D~1

C^1^1

C^^

C~2

C^2^1

C^

D^^

D~2

D~1

C^^2

后缀：^N，第N父版本；0表示自己。 1表示第一父版本，默认可以不输入；2表示第二个父版本——如果有。

后缀：~N，^重复N次； 0表示自己。

@ ： HEAD简写，指定当前分支的顶部。

# 常用命令

git clone 从其它仓库复制一份到本地。

git init 在当前目录中创建一个空git仓库，可以使用其它命令往仓库中添加内容。

git branch 分支管理命令:增、删、改、查。

git checkout 选择工作区使用的分支。

git reset 复位工作、暂区到初始状态。

git add 将修改放入到“暂存区”；

git commit 将修改提交到库中。

git log 查看版本提交日志。

git diff 查看不同版本之前的差异。

git push 将本仓库中的内容同步到其它仓库。

git fetch 将其它仓库更新到本仓库中

git pull 将其它仓库更新到李仓库中，并Merge到当前分支

git merge 将目标分支merge到当前分支

git rebase 改变当前分支拉出的位置，即在变更在新的起点上重新提交。

## git clone

git clone url/tecs

从服务器中复制一个版本库到本地，并检出默认的分支。

-b branch\_name 选项指定要检出的分支名。

-o repo\_name 可以指定库名，默认为“origin”。

## git branch

用于分支管理

git branch -v

查看：显示所有分支名，当前工作分支前面有“\*”。

git branch name start\_point

从start\_point位置创建一个新的分支；

start\_point不填时，从当前位置创建。

git branch –m old\_name new\_name

更改分支名。

git branch –D name

删除一个分支；

不能删除当前的分支。

git branch –u upstream branch\_name

为分支设置设置上游，在执行push, pull，reabse命令时，可使用该命令设置的分支信息。

## git checkout

用于切换工作区，使用不同的分支。

git checkout –b new\_branch\_name

创建一个新的分支并工作区切换到这个分支。

git checkout –B new\_branch\_name

作用与前一个命令样，如果分支已经存在将被覆盖。

git checkout –b branch\_name --track start\_point

创建一个分支并跟踪这个分支，即将这个分支设置为上游分支。

## git reset

git reset --mode commit

mode共有5种操作模式(选项)

--soft 将分支的指针复位到指定的版本;

--mix 将分支指针和暂存区复位到指定的版本；

--hard 将分支指针、斩存区、工作区复位到指定的版本。

--merge 同--hard, 保留工作区与暂存区的差异合并到新的工作区，冲突时操作失败。

--keep 同--hard,，保留工作区与HEAD的差异——合并到新的工作区，冲突时操作失败。

## git commit

将“暂存区”中的内容提交到仓库中，一些常用选项：

--amend：不新增一个版本，而直接替换当前版本。

-a：提交之前自动执行 git add 增加所有被修改的文件，新增文件还是需要手工不回。

-m “message”：添加注释，未执行时，将打开一个编辑器让用户输入。

--reset-author：重置作者信息。与--amend配合，可修改当前版本的作者。

## git log

显示日志信息，一些常用选项：

--oneline 以每个版本日志一行的方式显示

--decorate 在日志边上显示相关的引用信息（各分支、TAG指向的版本）

--graph 在日志之前用连线表示版本之前的关系

## git diff

git diff

默认是比较“工作区”与“暂存区”的差异。

git diff –cache

比较“暂存区”与当前分支的差异。

Git diff –canch new\_commit

比较“暂存区”与指定分支的差异。

git diff old\_commit new\_commit

比较两个不同版本的差异。

## git push

将本地的新的版本推送到远端中，默认只能添加新的版本，即只能以快进的方式将本地修改合并到远端。使用—force选项，可以进行任意的操作, 这是一个危险的选项，可能会导致已有的版本被覆盖（删除）。

## git fetch

将服务器中的修改同步到本地，默认获取所有远端中refs/heads/\*所有的分支、以及指向这些分支中版本的TAG。

git fetch orgin refs/heads/\*:refs/remotes/origin/\*

## git pull

从远端仓库中更新内容到本地，并将内容更新到当前分支。相当于git fetch 加上 git merge两个命令

## git merge

分支合并，将其它分支的修改内容合并到当前分支。默认选项下，没有冲突时，会自动完成合并；如果有冲突，会有提交，解决后需要手工提交。

可以将多个分支merge到将当前分支。

git merge --no-commit from\_branch\_xx

完成后使用 git commit来提交。

git merge --abort 放弃未完成的merge。

git merge --squash 只做merge动作，不记录merge信息，需要git commit来提交结果。

## git rebase

改变分支的拉出的位置。

git rebase --onto new\_base origin\_base head

将分支(origin\_base到 head这些修改)的起始位置从 origin\_base 移动到new\_base。Head可以是某个分支，不填时就当前分支；origin\_base省略时，使用当前分支的拉出位置。

-i 选项交互模式，可以选择每个版本的操作：丢弃，编辑注释，压缩（保留修改内容，丢弃日志）等

## git cherry-pick

对版本做一个patch，再合入到指定的分支，保留日志信息，但不保留版本之前的联系。

## git中svn 中的等价命令

svn checkout 等价于 git clone

svn update 等价于 git pull –r

svn commit –m “xxx” 等价于git commit –am “xxx” git push

# 常用的辅助命令

## git help

帮助系统，记不住命令没有关系，可以使用

git help

git help sub\_command

git command --help

## git config

git config

--global

选项整个Linux账号下都有效，属于用户的全局配置， 配置信息存储在~/.gitconfig文件中。

不使用这个选项则仅对当仓库有效，配置信息存储在当前仓库中的config文件中。

git config --global user.name “my\_name”

配置用户名

git config --global user.email [my\_emal@zte.com.cn](mailto:my_emal@zte.com.cn)

配置邮件地址

git config --global credential.helper store

配置保存 http方式使用的帐号与密码信息

git config --global credential.helper “cache –timeout=2600”

保存输入的帐号与密码信息一段时间

git config --global alias.your\_name “command args”

定义自己的命令，可以将常的选项包含进去，方便使用。

例如：gitconfig --global alias.l “log –decorate –graph”

## git tag

给指定版本打标签。

git tag tag\_name commit\_refs。

-a 选项，标签可以有作者日期等信息。

## git gc

重新整理打包库中的文件，同时会删除那些不可达的版本信息。使用git gc 可以减少磁盘使用，并提升后续操作性能。例如：分支被删除了，这个分支引用的版本都变成不可达，使用git gc 将会彻底删除这些节点。

## git stash

管理临时的存储区——保存当前工作区和暂存区的内容。

git stash save “message“

将当前工作区和暂存区中的内容存入“stash”中；

git stash list

列出所有已保存的清单；

git stash drop

删除；

git stash pop

恢复内容，并删除保存的内容；

git stash apply

恢复已保存的内容；

git stash clear

清除所有的已保存的内容。

## git show