## Name

git-fetch  - 从另一个存储库下载对象和引用

## 概要

git fetch [<options>] [<repository> [<refspec>…]]git fetch [<options>] <group>git fetch --multiple [<options>] [(<repository> | <group>)…]git fetch --all [<options>]

## 描述

从一个或多个其他存储库获取分支和/或标签（统称为“参考”），以及完成其历史记录所需的对象。远程跟踪分支已更新（请参阅下面的 <refspec> 描述以了解如何控制此行为）。

默认情况下，还会提取指向正在提取的历史记录的任何标记; 其效果是获取指向您感兴趣的分支的标记。可以使用 --tags 或 --no-tags 选项或通过配置远程。<name> .tagOpt 来更改此默认行为。通过使用显式提取标签的 refspec，您可以获取不指向您感兴趣的分支的标签。

git fetch可以从单个指定的存储库或 URL 中获取数据，也可以在给定 <group> 的情况下同时从多个存储库中获取，并且配置文件中存在远程。<group> 条目。（请参阅 git-config [1]）。

如果未指定远程，默认情况下origin将使用远程，除非为当前分支配置了上游分支。

被读取的引用的名称以及它们指向的对象名称将被写入.git/FETCH\_HEAD。这些信息可以被脚本或其他 git 命令使用，例如 git-pull [1]。

## 选项

--all

获取所有远端。

-a   --append

追加 refs 的 ref 名称和对象名称到现有的内容.git/FETCH\_HEAD。如果没有这个选项，旧的数据.git/FETCH\_HEAD将被覆盖。

--depth=<depth>

将提取限制为从每个远程分支历史记录的提示中指定的提交数量。如果获取到使用git clone和--depth=<depth>选项 创建的shallow存储库（请参阅 git-clone [1]），请将历史记录加深或缩短为指定的提交数。深化提交的标签不会被提取。

--deepen=<depth>

与 --depth 相似，只是它指定了来自当前浅层边界而不是每个远程分支历史记录的提示的提交数。

--shallow-since=<date>

加深或缩短浅储存库的历史记录以在 <date> 之后包含所有可访问的提交。

--shallow-exclude=<revision>

加深或缩短浅储存库的历史记录以排除可从指定远程分支或标记访问的提交。该选项可以多次指定。

--unshallow

如果源存储库已完成，请将浅层存储库转换为完整存储库，以消除浅存储库施加的所有限制。

如果源存储库较浅，则尽可能多地获取，以便当前存储库具有与源存储库相同的历史记录。

--update-shallow

默认情况下，从浅仓库中获取时，git fetch拒绝需要更新 .git / shallow 的引用。该选项更新 .git / shallow 并接受此类参考。

--dry-run

显示将做什么，不做任何改变。

-f   --force

当git fetch与<rbranch>:<lbranch> refspec 一起使用时，它拒绝更新本地分支，<lbranch>除非<rbranch>它获取的远程分支是后代<lbranch>。该选项将覆盖该检查。

-k   --keep

保持下载的包。

--multiple

允许指定几个 <repository> 和 <group> 参数。没有 <refspec>s 可能被指定。

-p   --prune

在提取之前，请删除远程不再存在的所有远程跟踪参考。如果标签仅由于默认标签自动跟随或由于 --tags 选项而被提取，则标签不会被修剪。但是，如果由于明确的 refspec（在命令行或远程配置中，例如，如果远程使用 --mirror 选项克隆）而获取标记，则它们也将受到修改。

-n   --no-tags

默认情况下，指向从远程存储库下载的对象的标签将在本地获取并存储。该选项将禁用此自动标记。远程的默认行为可以通过远程。<name> .tagOpt 设置指定。请参阅 git-config [1]。

--refmap=<refspec>

在获取命令行中列出的 ref 时，使用指定的 refspec（可以多次提供）将 ref 映射到远程跟踪分支，而不是remote.\*.fetch远程存储库的配置变量的值。有关详细信息，请参阅“配置远程跟踪分支”一节。

-t   --tags

从远程获取所有标签（即，将远程标签获取refs/tags/\*到具有相同名称的本地标签中），除此之外的任何其他内容都将被提取。即使 --prune 被使用（尽管如果标签也是明确的 refspec 的目的地，可能会被修剪，请参阅--prune），但单独使用此选项不会使标签遭到修剪。

--recurse-submodules=yes|on-demand|no

此选项控制是否以及在什么条件下也应提取填充的子模块的新提交。它可以用作布尔选项来完全禁用设置为no或设置为无条件递归到所有已填充的子模块时的递归yes，这是使用此选项时没有任何值的默认设置。使用on-demand仅递归到填充的子模块，当上层项目检索提交，更新子模块的参考一犯，是不是已经在本地子模块克隆。

-j   --jobs=<n>

用于提取子模块的并行子项的数量。每个子模块都将从不同的子模块中获取，从而获取多个子模块的速度会更快。默认情况下，子模块将一次取一个。

--no-recurse-submodules

禁用递归获取子模块（这与使用该--recurse-submodules=no选项具有相同的效果）。

--submodule-prefix=<path>

在信息消息（例如“获取子模块foo”）中打印路径前加上 <path> 。此选项在子模块上递归时用于内部。

--recurse-submodules-default=yes|on-demand

此选项在内部用于为 --recurse-submodules 选项临时提供非负的默认值。所有其他配置获取子模块递归的方法（例如 gitmodules [5]和 git-config [1]中的设置）将覆盖此选项，就像直接指定 --no-recurse 子模块一样。

-u   --update-head-ok

默认情况下git fetch拒绝更新对应于当前分支的头部。此标志禁用检查。这纯粹是供内部使用git pull来沟通git fetch，除非你正在实施自己的瓷器，否则你不应该使用它。

--upload-pack <upload-pack>

当给出并且从中获取的存储库被处理时git fetch-pack，--exec=<upload-pack>被传递给该命令以指定在另一端运行的命令的非默认路径。

-q   --quiet

通过 - 安静的 git-fetch-pack 和沉默任何其他内部使用的 git 命令。进度未报告给标准错误流。

-v   --verbose

详细。

--progress

当连接到终端时，默认情况下，标准错误流中会报告进度状态，除非指定了 -q 。即使标准错误流未定向到终端，此标志也会强制进度状态。

-4   --ipv4

仅使用 IPv4 地址，忽略 IPv6 地址。

-6   --ipv6

仅使用 IPv6 地址，忽略 IPv4 地址。

<repository>

作为提取或拉取操作源的“远程”存储库。此参数可以是 URL（请参阅下面的 GIT URLS 部分）或远程名称（请参阅下面的 REMOTES 部分）。

<group>

名称指的是存储库列表，作为配置文件中的远程数据 <group> 的值。（请参阅 git-config [1]）。

<refspec>

指定要获取哪些引用以及哪些本地引用要更新。当命令行中没有 <refspec> 时，remote.<repository>.fetch取而代之的是从变量中读取（参见下面的 CONFIGURED REMOTE-TRACKING BRANCHES）。

一个 <refspec> 参数的格式是一个可选的 plus +，后跟 source <src>，后跟一个冒号:，后跟目标 ref <dst> 。当 <dst>为空时，可以省略冒号。<src> 通常是 ref，但它也可以是拼写完整的十六进制对象名称。

tag <tag>意味着相同refs/tags/<tag>:refs/tags/<tag>; 它请求将所有内容都提取到给定的标签。

匹配 <src>的远程 ref 被取出，如果 <dst>不是空字符串，匹配它的本地引用使用 <src> 进行快速转发。如果使用可选的 plus +，即使不引起快速更新，本地 ref 也会被更新。

| **Note** | **When the remote branch you want to fetch is known to be rewound and rebased regularly, it is expected that its new tip will not be descendant of its previous tip (as stored in your remote-tracking branch the last time you fetched). You would want to use the + sign to indicate non-fast-forward updates will be needed for such branches. There is no way to determine or declare that a branch will be made available in a repository with this behavior; the pulling user simply must know this is the expected usage pattern for a branch.** |
| --- | --- |

## Git 网址

通常，URL 包含有关传输协议，远程服务器地址和存储库路径的信息。根据传输协议，这些信息可能不存在。

Git 支持 ssh，git，http 和 https 协议（另外，ftp 和 ftps 可用于提取，但效率低下，不推荐使用;不要使用它）。

本地传输（即 git：// URL）不进行身份验证，因此在不安全的网络上应谨慎使用。

以下语法可以与它们一起使用：

ssh://user@host.xz:port/path/to/repo.git/

git://host.xz:port/path/to/repo.git/

https://host.xz:port/path/to/repo.git/

ftps://host.xz:port/path/to/repo.git/

ssh 协议也可以使用另一种类似 scp 的语法：

user@host.xz：path / to / repo.git / 只有在第一个冒号前没有斜线时才能识别此语法。这有助于区分包含冒号的本地路径。例如，本地路径foo:bar可以被指定为绝对路径或./foo:bar避免被误解为 ssh url 。ssh 和 git 协议还支持〜用户名扩展：

ssh://user@host.xz:port/~user/path/to/repo.git/

git://host.xz:port/~user/path/to/repo.git/

user@host.xz:/~user/path/to/repo.git/

对于本地支持的本地存储库，可以使用以下语法：

/path/to/repo.git/

file:///path/to/repo.git/

这两种语法大多是相同的，除了克隆时，前者意味着 --local 选项。有关详细信息，请参阅 git-clone [1]。

当 Git 不知道如何处理某个传输协议时，它会尝试使用remote-<transport>远程助手（如果存在）。要显式请求远程助手，可以使用以下语法：

<transport> :: <address>其中<address>可能是一个路径，一个服务器和路径，或者是由被调用的特定远程助手识别的任意类似URL的字符串。有关详细信息，请参阅gitremote-helpers [1]。如果存在大量名称相似的远程存储库，并且您希望为它们使用不同的格式（例如，您使用的URL将被重写为可工作的URL），则可以创建一个形式的配置部分：[url] <actual url base>“] insteadOf = <other url base>例如，使用：[url：git：//git.host.xz/]] insteadOf = host。 xz：/ path / to / insteadOf = work：像“work：repo.git”或类似“host.xz：/path/to/repo.git”的URL将在任何需要URL为“混帐：//git.host。<repository> 论据：

Git 配置文件中的远程：$GIT\_DIR/config，

目录中的$GIT\_DIR/remotes文件，或

目录中的$GIT\_DIR/branches文件。

所有这些也允许你从命令行中省略 refspec，因为它们都包含 git 将默认使用的 refspec 。

### 在配置文件中命名远程

您可以选择提供您之前使用 git-remote [1]，git-config [1] 或者通过手动编辑$GIT\_DIR/config文件来配置的远程名称。此远程的 URL 将用于访问存储库。当您不在命令行中提供 refspec 时，将默认使用此远程程序的 refspec 。配置文件中的条目将如下所示：

        [remote "<name>"]

                url = <url>

                pushurl = <pushurl>

                push = <refspec>

                fetch = <refspec>

<pushurl>仅用于推动。它是可选的，默认为<url>。

### $GIT\_DIR/remotes中的命名文件

您可以选择提供文件的名称$GIT\_DIR/remotes。该文件中的 URL 将用于访问存储库。当你不在命令行上提供 refspec 时，这个文件中的 refspec 将被用作默认值。该文件应该具有以下格式：

        URL: one of the above URL format

        Push: <refspec>

        Pull: <refspec>

Push:行被使用git push，Pull:行被git pull和使用git fetch。可以为多个分支映射指定多个Push:和Pull:行。

### $GIT\_DIR/branches中的命名文件

您可以选择提供$GIT\_DIR/branches文件的名称。该文件中的 URL 将用于访问存储库。该文件应该具有以下格式：

        <url>#<head>

<url>是必须的; #<head>是可选的。

根据操作，如果你没有在命令行中提供一个参数，那么 git 将使用下列其中一个参考规范。<branch>这个文件中的名称$GIT\_DIR/branches和<head>默认master。

git fetch 使用：

        refs/heads/<head>:refs/heads/<branch>

git push 使用：

        HEAD:refs/heads/<head>

## 配置的远程跟踪分支

您经常通过定期和反复地从相同的远程存储库进行交互来进行交互。为了跟踪这种远程存储库的进度，git fetch可以配置remote.<repository>.fetch配置变量。

通常这样的变量可能看起来像这样：

[remote "origin"]

        fetch = +refs/heads/\*:refs/remotes/origin/\*

这种配置有两种使用方式：

在git fetch没有指定要在命令行上获取哪些分支和/或标签的情况下运行时（例如，git fetch origin或者git fetch，remote.<repository>.fetch值被用作refspecs），它们指定要获取哪些引用以及哪些本地引用要更新。上面的例子将获取存在于origin（即与值的左侧相匹配的任何 ref）中的所有分支，refs/heads/\*并更新refs/remotes/origin/\*层次结构中相应的远程跟踪分支。

当git fetch与明确的分支和/或标签运行在命令行，例如上取git fetch origin master中，<Refspec> s 上的命令行给出确定什么是要被取出（例如master在该示例中，这是一种短手master:，其反过来的意思是“取master分支，但我并没有明确说什么远程跟踪分支的命令行用它来更新”），示例命令将获取only的master分支。这些remote.<repository>.fetch值确定哪个远程跟踪分支（如果有的话）被更新。以这种方式使用时，这些remote.<repository>.fetch值在决定what获取时没有任何影响（即，在命令行列表 refspecs 时，这些值不会用作 refspecs）; 他们只是用来决定where 通过充当映射来存储被提取的 ref 。

remote.<repository>.fetch通过--refmap=<refspec>在命令行中提供参数，可以覆盖后面这些值的使用。

## 输出

“git fetch”的输出取决于所使用的传输方法; 本节描述通过 Git 协议（本地或通过 ssh）和 Smart HTTP 协议获取时的输出。

提取状态以表格形式输出，每行代表单个参考的状态。每一行的格式如下：

 <flag> <summary> <from> -> <to> [<reason>]

只有在使用 --verbose 选项时才会显示最新参考资料的状态。

在紧凑输出模式下，使用配置变量 fetch.output 指定，如果全部<from>或<to>在另一个字符串中找到，它将用\*另一个字符串替换。例如，master -> origin/master变成master -> origin/\*。

标志

一个字符表示 ref 的状态：

（空格）

成功取得快进;

+

成功强制更新;

-

为成功删除 ref；

t

为成功更新标签;

\*

为成功获取新的 ref ;

!

对于被拒绝或未能更新的裁判; 和

=

对于那些最新的并且不需要抓取的 ref 来说。

总结

对于成功获取的 ref，摘要以适合用作参数的形式git log（<old>..<new>在大多数情况下以及<old>...<new>强制非快进更新）显示 ref 的旧值和新值。

从

正在从中获取远程引用的名称，减去其refs/<type>/前缀。在删除的情况下，远程参考的名称是“（无）”。

至

正在更新的本地引用的名称，减去它的refs/<type>/前缀。

原因

一个可读的解释。在成功获取 ref 的情况下，不需要解释。对于失败的 ref，描述失败的原因。

## 例子

更新远程跟踪分支：$ git fetch origin 以上命令从远程 refs / heads / namespace 复制所有分支，并将它们存储到本地 refs / remotes / origin / 命名空间，除非使用分支。<name> .fetch 选项指定一个非默认的 refspec 。

明确使用 refspecs：

$ git fetch origin + pu：pu maint：tmp

这将更新（或根据需要创建）分支，pu和tmp通过从分支（分别）pu和maint远程存储库中获取来更新本地存储库。

pu是没有快进的，因为它是一个加号前缀，分支将被更新; tmp不会是。

查看远程分支，而不在本地存储库中配置远程程序：$ git fetch git：//git.kernel.org/pub/scm/git/git.git maint $ git log FETCH\_HEAD 第一个命令maint从存储库中获取分支在git://git.kernel.org/pub/scm/git/git.git和第二个命令使用FETCH\_HEAD用 git-log [1] 检查分支。抓取的对象最终会被 git 的内置管家删除（参见 git-gc [1]）。安全性抓取和推送协议并不旨在防止一方从另一个存储库窃取数据，而这些数据并不是要共享的。如果您需要保护私密数据免受恶意对等攻击，则最佳选择是将其存储在另一个存储库中。这适用于客户端和服务器。特别是，服务器上的名称空间对读取访问控制无效; 您应该只允许对读取访问整个存储库的用户可以信任的名称空间的读取权限。已知的攻击媒介如下：

victim 发送“有”线广告其具有的对象的 ID，这些对象的 ID 没有明确地意图共享，但是如果对等方也拥有它们，则可用于优化传送。攻击者选择一个对象 ID X 来窃取并向 X 发送一个 ref，但不需要发送 X 的内容，因为受害者已经拥有了它。现在受害者认为攻击者拥有 X，并且稍后将X的内容发送回攻击者。（这种攻击对于客户端来说在服务器上执行起来是最直接的，通过在客户端有权访问的命名空间中创建对 X 的引用，然后获取它，服务器在客户端上执行它的最可能方式是“合并“X 到一个公共分支，并希望用户在这个分支上做额外的工作，并将其推回服务器而不会注意到合并。）

和＃1一样，攻击者选择一个对象 ID X 来窃取。受害者发送攻击者已经拥有的对象 Y，并且攻击者错误地声称具有 X 而不是 Y，所以受害者发送 Y 作为与 X 相对的三角点。该三角点揭示 X 与攻击者类似的 X 的区域。

## Bugs

使用 --recurse 子模块只能在已签出的子模块中立即获取新的提交。例如，如果上游在超级项目的刚提取的提交中添加了新的子模块，则无法提取子模块本身，从而无法在稍后检查该子模块而无需再次执行提取。预计这将在未来的 Git 版本中得到修复。