**Git操作 - git log 命令的参数详解**

git log命令主要用于查看Git版本演变历史（也就是提交历史），同时根据追加的参数和选项不同，也会有不同的展示效果。

但默认git log命令显示出的x效果实在太丑，不好好打扮一下根本没法见人，打扮好了用alias命令拍个照片，就正式出道了！

**1、git log命令说明**

git log用于查询版本的历史，命令形式如下：

|  |
| --- |
| XML git log [<options>] [<since>..<until>] [[--] <path>...] |

默认的git log命令显示效果如下：



（之前文件详细说明过了，这里简单说一下，主要说明git log命令的参数）

**2、git log命令参数**

这条命令有很多参数选项，下面详细的说明：

**（1）不带参数**

1. 如果不带任何参数，它会列出所有历史记录，最近的排在最上方，显示提交对象的哈希值，作者、提交日期、和提交说明
2. 如果记录过多，则按上下键，Enter（向下显示一行），空格（显示下一页），来控制显示。
3. 按q键退出历史记录列表。

**（2）常用显示参数**

1. -p：按补丁格式显示每个更新的差异，比下一条--stat命令信息更全。
2. --stat：显示每次提交修改文件的统计信息，每个提交都列出了修改过的文件，以及其中添加和移除的行数，并在最后列出所有增减行数小计。
3. --shortstat：只显示--stat中最后的行数添加、修改、删除的统计。
4. --name-only：仅在提交信息后，显示已修改的文件清单。
5. --name-status：显示新增、修改、删除的文件清单。
6. --abbrev-commit：仅显示SHA-1校验和的前几个字符，而非所有的40个字符。
7. --relative-date：使用较短的相对时间，而不是完整格式显示日期（比如“2 weeks ago”）。
8. --graph：在日志旁以 ASCII 图形显示分支与合并历史。
9. --online：列表的形式查看历史版本记录，--pretty=oneline --abbrev-commit 合用的简写。
10. --pretty：使用其他格式显示历史提交信息。可用的选项包括oneline、short、full、fuller和 format（用来定义自己的格式）。

**（3）--pretty参数**

--pretty：自定义格式化显示历史提交信息。

选项有：

1）--pretty=oneline：一行显示，只显示哈希值和提交说明（--online本身也可以作为单独的属性）。

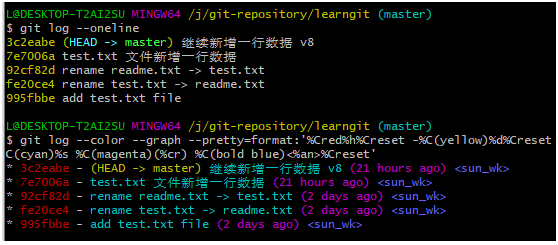
2）--pretty=format:” "：控制显示信息的格式。



**点击图片可查看完整电子表格**

3）--pretty=format:” "：控制显示信息的颜色。

1. 颜色格式：
2. *％Cred*：将颜色切换为红色
3. *％Cgreen*：将颜色切换为绿色
4. *％Cblue*：将颜色切换为蓝色
5. *%Creset*：空格
6. 举例：
7. 执行命令：git log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %C(cyan)%s %C(magenta)(%cr) %C(bold blue)<%an>%Creset' 两种效果如下：



1. 我们先进行断句分析：［%Cred%h］［%Creset -］［%C(yellow)%d ］［%Cblue%s］［%Cgreen(%cd)］［%C(bold blue)<%an>］ 然后就是很明显能得到的规律了:

* 写法：一个颜色＋一个内容，例如%Cred%h，依次排写。
* 颜色以%C开头，后边接几种颜色，还可以设置字体，如果要设置字体的话，要一块加个括号。 能设置的颜色值包括：reset（默认的灰色），normal，black，red，green，yellow，blue，magenta，cyan，white。（不能单独使用就加括号） 字体属性则有bold，dim，ul，blink，reverse。
* 内容可以是占位元字符，也可以是直接显示的普通字符（例如上面的-）

**（4）--date=参数**

--date=(relative|local|default|iso|rfc|short|raw)

仅对以人类可读格式显示的日期生效，例如使用--pretty.log.date这个config变量设置日志的默认值。

**--date有几个默认选项：**

1. --date=relative：显示相对于当前时间的日期，例如"2小时前"。
2. --date=local：显示用户本地时区中的时间戳。
3. --date=iso或--date=iso8601：以ISO 8601格式显示时间戳。
4. --date=rfc或--date=rfc2822：以RFC 2822格式显示时间戳，通常在电子邮件中找到。
5. --date=short：仅以YYYY-MM-DD格式显示日期，而不显示时间。
6. --date=raw：以内部原始Git格式%s %z格式显示日期。
7. --date=default：显示原始时区(提交者或作者)的时间戳。

**也可以自定义格式（需要git版本2.6.0以上）**：

比如：--date=format:'%Y-%m-%d %H:%M:%S'，会格式化成：2016-01-13 11:32:13。

其他的格式化占位符如下：

1. %a：星期的缩写。
2. %A：星期的全名。
3. %b：月份的缩写。
4. %B：月份的全称。
5. %c：适用于区域设置的日期和时间表示。
6. %d：月中的天作为十进制数字（01 – 31）。
7. %H： 24小时制的小时（00 – 23）。
8. %I：2小时格式的小时（01 – 12）。
9. %j：一年中的天作为十进制数字（001 – 366）。
10. %m：以十进制数字表示的月份（01 – 12）。
11. %M：分钟以十进制数字表示（00 – 59）。
12. %p：当前语言环境的"上午/下午"，12小时制的指示器。
13. %S：秒作为十进制数字（00 – 59）。
14. %U：一年中的周为十进制数字，周日为一周的第一天（00 – 53）。
15. %w：工作日为十进制数字（0 – 6；星期日为0）。
16. %W：一年中的星期作为十进制数字，星期一作为星期的第一天（00 – 53）
17. %x：当前语言环境的日期表示。
18. %X：当前语言环境的时间表示。
19. %y：无世纪的年份，为十进制数字（00 – 99），也就是年份没有前两位。
20. %Y：带世纪的年份，以十进制数表示。
21. %z，%Z：时区名称或时区缩写，取决于注册表设置； 如果时区未知，则没有字符。
22. %%：表示百分号。

**（5）筛选参数**

**1）按数量**

* -n：显示前n条提交信息。

**2）按日期**

1. --after=
2. 比如git log --after="2014-7-1”，显示2014年7月1号之后的commit(包含7月1号)。
3. 后边的日期还可以用相对时间表示，比如"1 week ago"和”yesterday"，比如git log --after="yesterday"。
4. --before=：同上。

另外这两条命令可以同时使用表示时间段，比如git log --after="2014-7-1" --before="2014-7-4"

同理：--since、--until和--after、--before是一个意思。

--since, --after：仅显示指定时间之后的提交。

--until, --before：仅显示指定时间之前的提交。

**3）按作者和提交者**

--author=：仅显示作者匹配指定字符串的提交。

1. 比如git log --author=“John"，显示John贡献的commit。
2. 注意：作者名不需要精确匹配，只需要包含就行了。
3. 而且可以使用正则表达式，比如git log --author="John|Mary”，搜索Marry和John贡献的commit。
4. 而且这个--author不仅包含作者名，还包含作者的email，所以你可以用这个搜索email。

--committer：仅显示提交者匹配指定字符串的提交。

**4）按commit描述**

--grep=：仅显示提交说明中包含指定字符串的提交。

1. 比如：git log --grep="JRA-224"
2. 而且可以传入-i用来忽略大小写。
3. 注意：如果想同时使用--grep和--author，必须在附加一个--all-match参数。

**5）按文件**

--（空格）文件或--［后边没有文件］

1. 有时你可能只对某个文件的修改感兴趣，或你只想查看跟某个文件相关的历史信息，你只需要插入你感兴趣文件的路径，［对，是路径，所以经常是不太好用］就可以了。
2. 比如：git log -- foo.py bar.py ，只返回和foo.py文件或bar.py文件的相关commit。
3. 这里的--是告诉Git，后面的参数是文件路径而不是branch（分支）的名字。 如果后面的文件路径不会和某个branch产生混淆，你可以省略--，比如git log foo.py。
4. 另外，后边的路径还支持正则，比如：git log \*install.md ，是指定项目路径下的所有以install.md结尾文件的提交历史。
5. 另外，文件名应该放到参数的最后位置，通常在前面加上--并用空格隔开表示是文件。
6. 另外，git log file/，可以查看file文件夹下所有文件的提交记录。

**6）按分支**

--：

1. --branchName branchName参数：为任意一个分支名字，查看某个分支上的提交记录。
2. 需要放到参数中的最后位置处。
3. 如果分支名与文件名相同，系统会提示错误，可通过--选项，来指定给定的参数，是分支名还是文件名。
4. 比如：在当前分支中有一个名为v1的文件，同时还存在一个名为v1的分支。如下：
5. git log v1 --，此时的v1代表的是分支名字（--后边是空的）。
6. git log -- v1，此时的v1代表的是名为v1的文件。
7. git log v1 -- v1：代表v1分支下的v1文件。

**7）按内容**

-S"<string>"、-G"<string>"：仅显示添加或删除内容匹配指定字符串的提交。

1. 有时你想搜索，新增或删除某行代码相关的commit. 可以使用这条命令。
2. 假设你想知道Hello,World!这句话，是什么时候加入到项目里去的，可以用：git log -S"Hello,World!"。
3. 另外，如果你想使用正则表达式去匹配，而不是字符串，那么你可以使用-G代替-S。
4. 这是一个非常有用的debug工具，使用他可以定位所有跟某行代码相关的commit，甚至可以查看某行是什么时候被copy的，什么时候移到另外一个文件中去的。
5. 注意：-S后没有"="，与查询内容之间也没有空格符。

**8）按范围**

git log <since>..<until>：

1. 这个命令可以查看某个范围的commit。
2. 这个命令非常有用当你使用branch做为range参数的时候. 能很方便的显示2个branch（分支）之间的不同。
3. 比如：git log master..feature，master..feature这个range（范围），包含了在feature分支存在，而在master分支没有的所有commit。同样，如果是feature..master，意思是包含所有master分支有，但是feature分支没有的commit。
4. 另外，如果是三个点，表示或的意思：git log master...test，意思是查询master或test分支中的提交记录。

**9）过滤掉merge commit**

--no-merges：

1. 默认情况下git log会输出merge commit（合并提交）。你可以通过--no-merges标记来过滤掉merge commit，例如：git log --no-merges。
2. 另外，如果你只对merge commit感兴趣可以使用--merges参数，例如：git log --merges。

**10）按标签tag**

git log v1.0：

1. 直接这样是查询标签之前的commit。
2. 加两个点，例如：git log v1.0..，是查询从v1.0以后的提交历史记录(不包含v1.0)。

**11）按commit**

1. git log commit：查询commit之前的记录，包含commit。
2. git log commit1 commit2：查询commit1与commit2之间的记录，包括commit1和commit2。
3. git log commit1..commit2：同上，但是不包括commit1。

说明：

* 其中，commit可以是提交哈希值的简写模式，也可以使用HEAD代替。
* HEAD代表最后一次提交，HEAD^为最后一个提交的父提交，等同于HEAD~1。
* HEAD~2代表倒数第二次提交。

|  |
| --- |
| 提示：Git的指令是非常丰富的，通过git help log 或者git help --web log，可以以浏览器的方式查看更多关于log的指令。  参考：   * [https://www.cnblogs.com/bellkosmos/p/5923439.html](https://links.jianshu.com/go?to=https%3A%2F%2Fwww.cnblogs.com%2Fbellkosmos%2Fp%2F5923439.html) * [https://git-scm.com/book/zh/v2/](https://links.jianshu.com/go?to=https%3A%2F%2Fgit-scm.com%2Fbook%2Fzh%2Fv2%2F) |

作者：繁华似锦Fighting 链接：https://www.jianshu.com/p/19da385b288d 来源：简书 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。