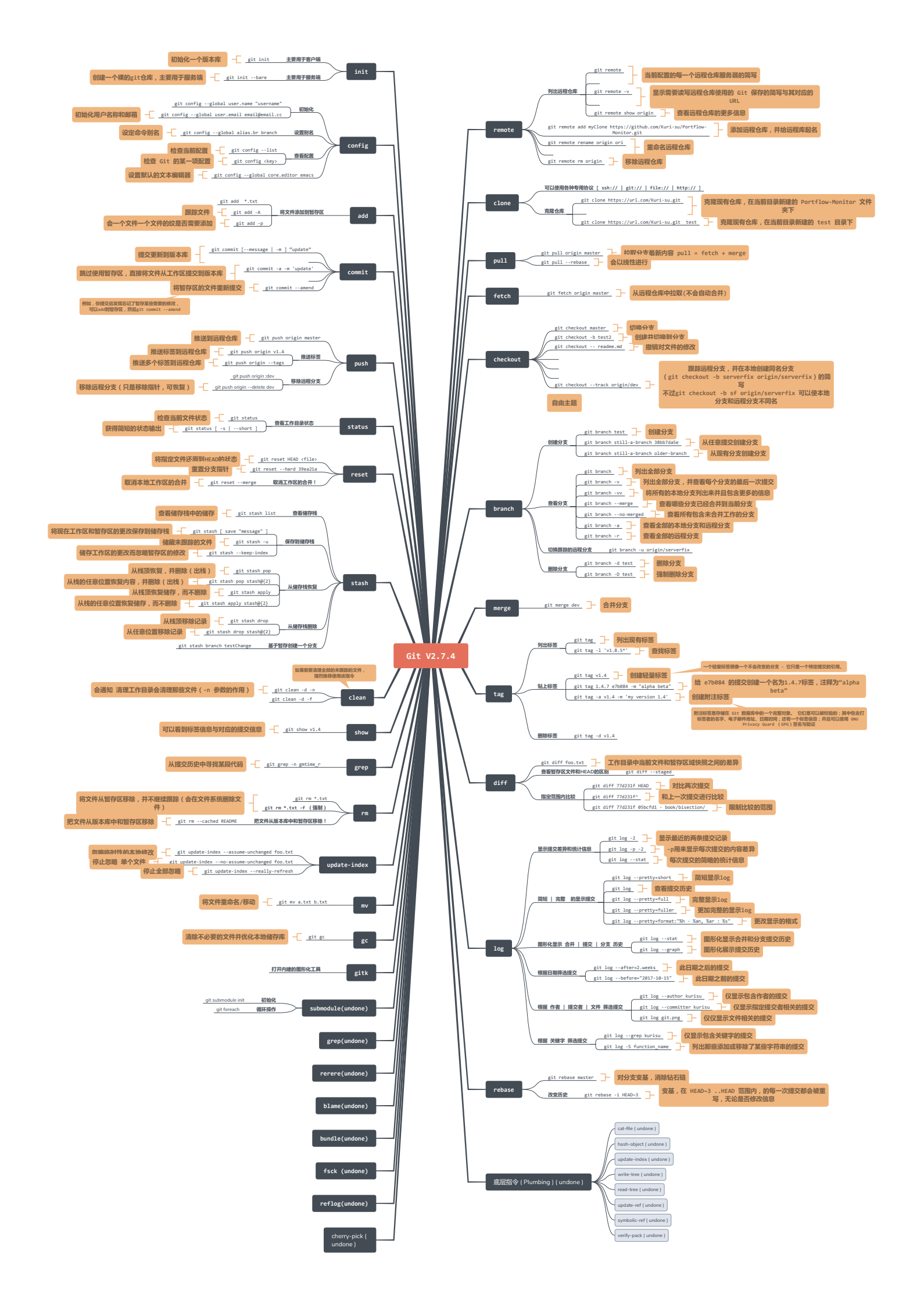
Git V2.7.4
init
初始化一个版本库
git init
主要用于客户端
创建一个裸的git仓库，主要用于服务端
git init --bare
主要用于服务端
config
git config --global user.name "username"
初始化用户名称和邮箱
git config --global user.email email@email.cc
初始化
设定命令别名
git config --global alias.br branch
设置别名
检查当前配置
git config --list
检查 Git 的某一项配置
git config <key>
查看配置
设置默认的文本编辑器
git config --global core.editor emacs
add
git add *.txt
跟踪文件
git add -A
将文件添加到暂存区
会一个文件一个文件的纹是否需要添加
git add -p
commit
提交更新到版本库
git commit [--message | -m] "update"
跳过使用暂存区，直接将文件从工作区提交到版本库
git commit -a -m 'update'
将暂存区的文件重新提交
git commit --amend
例如，你提交后发现忘记了暂存某些需要的修改，可以add到暂存区，然后git commit --amend
push
推送到远程仓库
git push origin master
推送标签到远程仓库
git push origin v1.4
推送标签
推送多个标签到远程仓库
git push origin --tags
git push origin :dev
移除远程分支
移除远程分支（只是移除指针，可恢复）
git push origin --delete dev
status
检查当前文件状态
git status
查看工作目录状态
获得简短的状态输出
git status [-s | --short]
reset
将指定文件还原到HEAD的状态
git reset HEAD <file>
重置分支指针
git reset --hard 39ea21a
取消本地工作区的合并
git reset --merge
取消工作区的合并！
stash
查看储存栈中的储存
git stash list
查看储存栈
将现在工作区和暂存区的更改保存到储存栈
git stash [save "message"]
保存到储存栈
储藏未跟踪的文件
git stash -u
储存工作区的更改而忽略暂存区的修改
git stash --keep-index
从栈顶恢复，并删除（出栈）
git stash pop
从栈的任意位置恢复内容，并删除（出栈）
git stash pop stash@{2}
从储存栈恢复
从栈顶恢复储存，而不删除
git stash apply
从栈的任意位置恢复储存，而不删除
git stash apply stash@{2}
从栈顶移除记录
git stash drop
从储存栈删除
从任意位置移除记录
git stash drop stash@{2}
git stash branch testChange
基于暂存创建一个分支
clean
如果需要清理全部的未跟踪的文件，强烈推荐使用该指令
会遇知 清理工作目录会清理那些文件（-n 参数的作用）
git clean -d -n
git clean -d -f
show
可以看到标签信息与对应的提交信息
git show v1.4
grep
从提交历史中寻找某段代码
git grep -n gmtime_r
rm
将文件从暂存区移除，并不继续跟踪（会在文件系统删除文件）
git rm *.txt
git rm *.txt -f （强制）
把文件从版本库中和暂存区移除！
把文件从版本库中和暂存区移除
git rm --cached README
update-index
忽略临时性的本地修改
git update-index --assume-unchanged foo.txt
停止忽略 单个文件
git update-index --no-assume-unchanged foo.txt
停止全部忽略
git update-index --really-refresh
mv
将文件重命名/移动
git mv a.txt b.txt
gc
清除不必要的文件并优化本地储存库
git gc
gitk
打开内建的图形化工具
submodule(undone)
git submodule init
初始化
git foreach
循环操作
grep(undone)
rerere(undone)
blame(undone)
bundle(undone)
fsck (undone)
reflog(undone)
cherry-pick (undone)
remote
列出远程仓库
git remote
当前配置的每一个远程仓库服务器的简写
git remote -v
显示需要读写远程仓库使用的 Git 保存的简写与其对应的 URL
git remote show origin
查看远程仓库的更多信息
git remote add myClone https://github.com/Kuri-su/Portflow-Monitor.git
添加远程仓库，并给远程库起名
git remote rename origin ori
重命名远程仓库
git remote rm origin
移除远程仓库
clone
可以使用各种专用协议 [ssh:// | git:// | file:// | http://]
克隆仓库
git clone https://url.com/Kuri-su.git
克隆现有仓库，在当前目录新建的 Portflow-Monitor 文件夹下
git clone https://url.com/Kuri-su.git test
克隆现有仓库，在当前目录新建的 test 目录下
pull
git pull origin master
拉取分支最新内容 pull = fetch + merge
git pull --rebase
会以线性进行
fetch
git fetch origin master
从远程仓库中拉取(不会自动合并)
checkout
git checkout master
切换分支
git checkout -b test2
创建并切换到分支
git checkout -- readme.md
撤销对文件的修改
git checkout --track origin/dev
跟踪远程分支，并在本地创建同名分支
（git checkout -b serverfix origin/serverfix）的简写
不过git checkout -b sf origin/serverfix 可以使本地分支和远程分支不同名
自由主题
branch
创建分支
git branch test
创建分支
git branch still-a-branch 38bb7da5e
从任意提交创建分支
git branch still-a-branch older-branch
从现有分支创建分支
查看分支
git branch
列出全部分支
git branch -v
列出全部分支，并查看每个分支的最后一次提交
git branch -vv
将所有的本地分支列出来并且包含更多的信息
git branch --merge
查看哪些分支已经合并到当前分支
git branch --no-merged
查看所有包含未合并工作的分支
git branch -a
查看全部的本地分支和远程分支
git branch -r
查看全部的远程分支
切换跟踪的远程分支
git branch -u origin/serverfix
删除分支
git branch -d test
删除分支
git branch -D test
强制删除分支
merge
git merge dev
合并分支
tag
列出标签
git tag
列出现有标签
git tag -l 'v1.8.5*'
查找标签
贴上标签
git tag v1.4
创建轻量标签
一个轻量标签很像一个不会改变的分支 - 它只是一个特定提交的引用。
git tag 1.4.7 e7b084 -m "alpha beta"
给 e7b084 的提交创建一个名为1.4.7标签，注释为“alpha beta”
git tag -a v1.4 -m 'my version 1.4'
创建附注标签
附注标签是存储在 Git 数据库中的一个完整对象。 它们是可以被校验的；其中包含打标签者的名字、电子邮件地址、日期时间；还有一个标签信息；并且可以使用 GNU Privacy Guard （GPG）签名与验证
删除标签
git tag -d v1.4
diff
git diff foo.txt
工作目录中当前文件和暂存区域快照之间的差异
查看暂存区文件和HEAD的区别
git diff --staged
指定范围内比较
git diff 77d231f HEAD
对比两次提交
git diff 77d231f^
和上一次提交进行比较
git diff 77d231f 05bcfd1 - book/bisection/
限制比较的范围
log
显示提交差异和统计信息
git log -2
显示最近的两条提交记录
git log -p -2
-p用来显示每次提交的内容差异
git log --stat
每次提交的简略的统计信息
简短 | 完整 的显示提交
git log --pretty=short
简短显示log
git log
查看提交历史
git log --pretty=full
完整显示log
git log --pretty=fuller
更加完整的显示log
git log --pretty=format:"%h - %an, %ar : %s"
更改显示的格式
图形化显示 合并 | 提交 | 分支 历史
git log --stat
图形化显示合并和分支提交历史
git log --graph
图形化展示提交历史
根据日期筛选提交
git log --after="2.weeks
此日期之后的提交
git log --before="2017-10-15"
此日期之前的提交
根据 作者 | 提交者 | 文件 筛选提交
git log --author kurisu
仅显示包含作者的提交
git log --committer kurisu
仅显示指定提交者相关的提交
git log git.png
仅仅显示文件相关的提交
根据 关键字 筛选提交
git log --grep kurisu
仅显示包含关键字的提交
git log -S function_name
列出那些添加或移除了某些字符串的提交
rebase
git rebase master
对分支变基，消除钻石链
改变历史
git rebase -i HEAD~3
变基，在 HEAD~3 ..HEAD 范围内，的每一次提交都会被重写，无论是否修改信息
底层指令 (Plumbing) (undone)
cat-file (undone)
hash-object (undone)
update-index (undone)
write-tree (undone)
read-tree (undone)
update-ref (undone)
symbolic-ref (undone)
verify-pack (undone)