- 简介
- 配置
- 文件状态
- 命令分析
    - git status
    - git commit
    - git checkout
- 常用命令
    - 工作区暂存区文件管理
    - 比对文件
    - 提交文件
    - 工作日志
    - 分支
    - 远端
    - 配置
    - 其他
- skill
    - 忽略文件/目录
    - 提交空目录
    - 别名
- 速查表


```
$ hexo n post git-note
```

```
title: git笔记
tags: git
```

简介

部分重要概念。

- HEAD
- Working Directory / Repository / Stage
- branch
- remote
- tag

配置

1. 打开Git Bash，配置用户名字和邮件地址

```
$ git config  --global user.name "Your Name"
$ git config --global user.email "email@example.com"
```

1. 本机创建SSH，Github配置SSH

一路回车

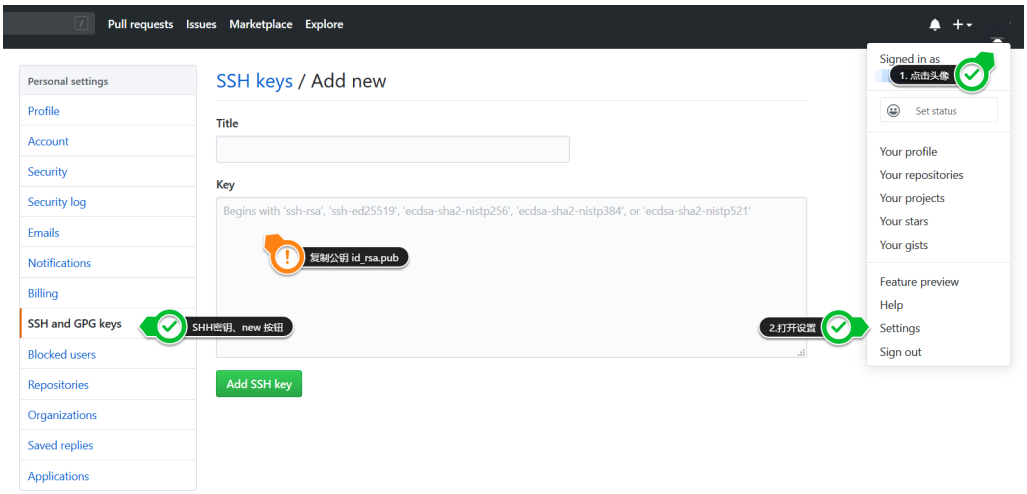
```
$ ssh-keygen -t rsa -C "youremail@example.com"
```

win+r输入%USERPROFILE%/.ssh,成功打开且有以下文件则成功.

1. id_ras：私钥，不能泄露
2. id_ras.pub：公钥，可以告诉任何人

以下操作需要Github账号，若无需注册。

添加SSH密钥 传送门



1. 测试配置
   - Github新建仓库，初始化
   - 本地clone下来，修改后提交，查看提交者信息是否正确
   - 推送至远端仓库，输入密码
   - 再次修改后提交，若SHH配置成功，无须输入密码

操作省略..

文件状态

简写	英文	翻译
M	modified	修改
R	renamed	重命名
C	both modified	冲突
R	Untracked	未跟踪

命令分析

一些常用的命令分析，我很喜欢命令行，不过source tree也挺方便的。

git status

输出信息

- 当前分支
- 远端分支状态（是否拉取/更新）
- 暂存区

- 工作区

实例分析

```
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   src/main.js

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   src/components/person-filed/index.vue
        modified:   src/router.js
        modified:   src/views/loop-action/index.vue

# ===== 翻译分割线 =====

位于master分支
您的分支超前“origin / master”一次提交。
  （使用“git push”发布您的本地提交）

要提交的更改：
  （使用“git reset HEAD <file> ...”取消暂存）

        修改：src / main.js

未提交更改的更改：
  （使用“git add <file> ...”来更新将要提交的内容）
  （使用“git checkout  -  <file> ...”来丢弃工作目录中的更改）

        修改：src / components / person-filed / index.vue
        修改：src / router.js
        修改：src / views / loop-action / index.vue
```

git commit

输出信息

- 提交分支
- commit hash值
- 修改文件数量

- 增删行数

实例分析

```
[master 2918d65] 1
 1 file changed, 1 insertion(+)

# ===== 翻译分割线 =====

 [master 2918d65] 1
  1个文件已更改，1行插入（+）
```

git checkout

自修改后还没有被放到暂存区--回到版本库的状态；
已经添加到暂存区后，又作了修改--就回到添加到暂存区后的状态。
git checkout其实是用版本库里的版本替换工作区的版本，
无论工作区是修改还是删除，都可以“一键还原”。

常用命令

个人常用命令，整理到一块，以后忘记了就直接复制，省事也方便。

工作区暂存区文件管理

```
# 清除工作区指定路径(<path>)下的所有文件修改（重置文件）
$ git checkout -- <path>

# 清空暂存区指定路径(<path>)文件（不重置修改）
$ git reset -- <path>

# 清空工作和暂存区的所有更改（重置本次提交，不会处理untracked files）
$ git reset HEAD --hard

# 删除 untracked files(-f) 包括目录(-d)
$ git clean -fd
```

比对文件

```
# 比对指定路径（<path>）文件和暂存区的区别
$ git diff <commit> -- <path>

# 比对已经暂存起来的文件(staged)和上次提交时的快照之间(HEAD)的差异
$ git diff --cached
$ git diff --staged

# 比对指定路径（<path>）两次提交
$ git diff <hash1> <hash2> -- <path>
```

```
# 比对行改动，不显示具体内容
$ git diff --stat
```

提交文件

```
# 全部暂存并提交
$ git commit -am "commit log"

# 多行插入空行提交
$ git commit -m '1.line-1' - m '2.line-2'

# 多行提交
$ git commit -m '
1. line-1
2. line-2
'
```

工作日志

```
# 简化工作日志
$ git log --pretty=oneline

# 查看分支合并情况
$ git log --graph

# 查看分支合并情况，简化提交信息、hash简写
$ git log --graph --pretty=oneline --abbrev-commit

# 美化输出、查看分支合并情况、简化提交信息、hash简写
$ git log --color --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset'  --graph  --abbrev-commit
```

分支

```
# 重命令分支
$ git branch -m <old_name> <new_name>

# 切换分支
$ git checkout <branch>

# 合并分支
$ git merge <branch>
```

远端

```
# 远端版本信息
$ git remote -v

# 添加git远端仓库
$ git remote add <url>

# 拉取远端分支提交
$ git pull origin master

# 推送远端分支提交
$ git push origin master

# 拉取远端分支到本地新分支
$ git checkout -b <new_branch> <remote> <branch>
```

配置

```
# 查看全局配置列表
$ git config --global --list

# 查看本地仓库配置列表
$ git config --local --list
```

其他

```
# 命令历史
$ git reflog

# 变基
$ git rebase -i
```

skill

一些简简单单的小技巧。

忽略文件/目录

根目录创建.gitignore文件。

windows系统需使用命令行创建，打开cmd，定位。

```
> type nul > .gitignore
```

.gitignore文件添加需要忽略的文件/目录即可。

一般不需要自己编辑，github官方也提供了不同语言的.gitignore 传送门

提交空目录

创建 .gitkeep 文件，内容如下

```
# Ignore everything in this directory
*
# Except this file !.gitkeep
```

别名

GIt 支持为命令自定义别名，比如我们希望**全局设置** `git br` 映射为 `git branch`，**仓库设置** `git st` 映射为 `git status`，我们可以在终端这样配置。

```
# 配置别名
git config --global alias.br branch
git config --local alias.st status
```

现在就可以使用`git br` 、`git st`了，不过`git st`是**仓库级别**的设置，切换到其他仓库就无效了。我不需要省敲几个键，这样子的映射对我无效，我需要映射的是一些很长难输入又实用的命令。我们先删除它，再配置我自己偏好的别名。

```
# 删除别名
git config --global --unset alias.br
git config --local --unset alias.st
```

我们本地还有全部的别名都被删除了，当然你也可以直接修改配置文件，但是不推荐。

```
git config --global alias.logs "log --color --graph --pretty=format:'%Cred%h%Creset -
%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"
git config --global alias.detail-log "log --color --graph --pretty=format:'%Cred%h%Creset -
%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit"
```

现在`git logs` 还有`git detail-log`都可以删除漂亮的git日志 :-)

速查表

Git 常用命令速查表

master :默认开发分支　　Head :默认开发分支
origin :默认远程版本库　　Head^ :Head 的父提交

创建版本库

```
$ git clone <url>              #克隆远程版本库
$ git init                     #初始化本地版本库
```

修改和提交

```
$ git status                   #查看状态
$ git diff                     #查看变更内容
$ git add .                    #跟踪所有改动过的文件
$ git add <file>               #跟踪指定的文件
$ git mv <old> <new>           #文件改名
$ git rm <file>                #删除文件
$ git rm --cached <file>       #停止跟踪文件但不删除
$ git commit -m "commit message"
                               #提交所有更新过的文件
$ git commit --amend           #修改最后一次提交
```

查看提交历史

```
$ git log                      #查看提交历史
$ git log -p <file>            #查看指定文件的提交历史
$ git blame <file>             #以列表方式查看指定文件
                               的提交历史
```

撤消

```
$ git reset --hard HEAD        #撤消工作目录中所有未提交
                               文件的修改内容
$ git checkout HEAD <file>     #撤消指定的未提交文件的修
                               改内容
$ git revert <commit>          #撤消指定的提交
```

分支与标签

```
$ git branch                   #显示所有本地分支
$ git checkout <branch/tag>    #切换到指定分支或标签
$ git branch <new-branch>      #创建新分支
$ git branch -d <branch>       #删除本地分支
$ git tag                      #列出所有本地标签
$ git tag <tagname>            #基于最新提交创建标签
$ git tag -d <tagname>         #删除标签
```

合并与衍合

```
$ git merge <branch>           #合并指定分支到当前分支
$ git rebase <branch>          #衍合指定分支到当前分支
```

远程操作

```
$ git remote -v                #查看远程版本库信息
$ git remote show <remote>     #查看指定远程版本库信息
$ git remote add <remote> <url>
                               #添加远程版本库
$ git fetch <remote>           #从远程库获取代码
$ git pull <remote> <branch>   #下载代码及快速合并
$ git push <remote> <branch>   #上传代码及快速合并
$ git push <remote> :<branch/tag-name>
                               #删除远程分支或标签
$ git push --tags              #上传所有标签
```

Git Cheat Sheet <CN> (Version 0.1)　　# 2012/10/26　-- by @riku　< riku@gitcafe.com / http://riku.wowubuntu.com >

<!--

- Git log 高级用法
- Git config 配置
- Git diff
- Git使用中的一些奇技淫巧

-->