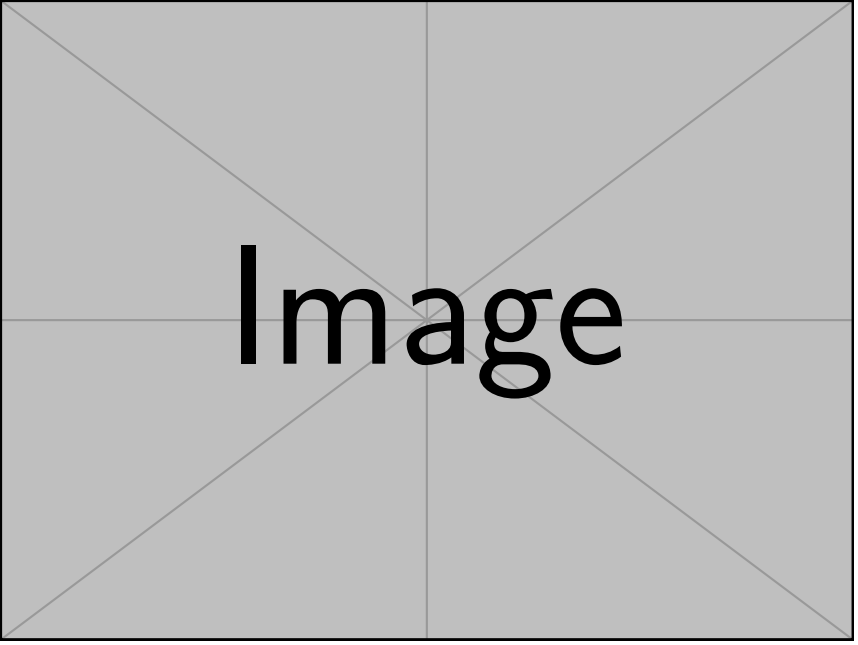


科研工具介绍

作者：张鑫

时间：2023/1/8

目录

前言

How do you work in Quarto? You can use whichever tool you're comfortable with (RStudio, Jupyter, GitHub, VS Code, etc). Developing your quarto site will have the same basic workflow, no matter which tool you use. It is very iterative, and each is explored more below. mystyleHow do you work in Quarto? You can use whichever tool you're comfortable with (RStudio, Jupyter, GitHub, VS Code, etc). Developing your quarto site will have the same basic workflow, no matter which tool you use. It is very iterative, and each is explored more below. mystyle

第 1 章 Mac 电脑设置

1.1 安装 Command Line Tools

```
xcode-select --install
```

1.2 Git 入门

1.2.1 Github 使用

1.2.2 安装 Git

1.2.3 Git 配置

安装完成后，还需要最后一步设置，在命令行输入：

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

1.2.4 本地仓库

a. git init 在本地创建一个 Git 仓库；
b. git add . 将项目添加到暂存区；
c. git commit -m “注释内容” 将项目提交到 Git 仓库；

1.2.5 远程仓库 Github

1.2.5.1 配置 SSH (参考通过 SSH 连接到 GitHub)

Github 使用 SSH 配置，初始需要以下三个步骤

1. 使用秘钥生成工具生成 rsa 秘钥和公钥
2. 将 rsa 公钥添加到代码托管平台
3. 将 rsa 秘钥添加到 ssh-agent 中，为 ssh client 指定使用的秘钥文件

具体操作如下：

第一步：检查本地主机是否已经存在 ssh key

```
cd ~/.ssh
ls ## 看是否存在 id_rsa 和 id_rsa.pub文件，如果存在，说明已经有SSH Key
```

如下图所示，则表明已经存在

```
admin@DESKTOP-F25R12Q MINGW64 ~/Desktop
$ cd ~/.ssh

admin@DESKTOP-F25R12Q MINGW64 ~/.ssh
$ ls
id_rsa  id_rsa.pub  known_hosts
```

如果存在，直接跳到第三步

第二步：生成 ssh key 如果不存在 ssh key，使用如下命令生成

```
ssh-keygen -t rsa -C "xxx@xxx.com"
//执行后一直回车即可
```

生成完以后再用第二步命令，查看 ssh key

第三步：获取 ssh key 公钥内容（id_rsa.pub）

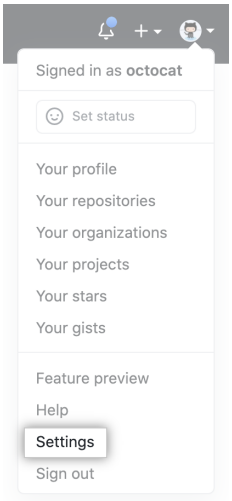
```
cd ~/.ssh
cat id_rsa.pub
```

如下图所示，复制该内容

```
$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDtI4vjeQ9Rryj/pVQHvNj+2xBQxcpuK6yw9haS6SQL
BPk3MT1h2BzoTaQ4XeJa30KbYwEuDTvNTdgAO2829+NCogOo8q9O2+/hwOC/nY9K5pafRStXOpjyzxgC
obem7boWXC2zUx5xXnvNfKAO7AhftCo4mrgITHoXUn2BsmO8Mbzj3p8btQmHqEgrO+paX4Ay//OkCH56
wj+5TCfbccNuLyH3P+5wxNjOco7dndGEJ9aORRtWzfa/GyQfysv4p1p5YyZ2Haj0muY2TyrbO7G+QQay
g6y7XqGzUZQMmwnwOQTuG6WvNtaY1ulXXRXjL34mdXKOCSOdOPzCiqKUc/e7 v_tigachen@tencent.
com
```

第四步：Github 账号上添加公钥

进入 Settings 设置



添加 ssh key，把刚才复制的内容粘贴上去保存即可

第五步：验证是否设置成功

```
ssh -T git@github.com
```

显示如下信息表明设置成功

```
$ ssh -T git@github.com
Hi jiahao12121! You've successfully authenticated, but GitHub does not provide s
hell access.
```

设置成功后，即可不需要账号密码 clone 和 push 代码

注意之后在 clone 仓库的时候要使用 ssh 的 url，而不是 https！

1.2.6 验证原理

SSH 登录安全性由非对称加密保证，产生密钥时，一次产生两个密钥，一个公钥，一个私钥，在 git 中一般命名为 id_rsa.pub, id_rsa。

那么如何使用生成的一个私钥一个公钥进行验证呢?

本地生成一个密钥对，其中公钥放到远程主机，私钥保存在本地

当本地主机需要登录远程主机时，本地主机向远程主机发送一个登录请求，远程收到消息后，随机生成一个字符串并用公钥加密，发回给本地。本地拿到该字符串，用存放在本地的私钥进行解密，再次发送到远程，远程比对该解密后的字符串与源字符串是否等同，如果等同则认证成功。

1.2.7 通俗解释！！

重点来了：一定要知道 ssh key 的配置是针对每台主机的！，比如我在某台主机上操作 git 和我的远程仓库，想要 push 时不输入账号密码，走 ssh 协议，就需要配置 ssh key，放上去的 key 是当前主机的 ssh 公钥。那么如果我换了一台其他主机，想要实现无密登录，也就需要重新配置。

下面解释开头提出的问题：

（1）为什么要配?

配了才能实现 push 代码的时候不需要反复输入自己的 github 账号密码，更方便

（2）每使用一台主机都要配?

是的，每使用一台新主机进行 git 远程操作，想要实现无密，都需要配置。并不是说每个账号配一次就够了，而是每一台主机都需要配。

（3）配了为啥就不用密码了?

因为配置的时候是把当前主机的公钥放到了你的 github 账号下，相当于当前主机和你的账号做了一个关联，你在这台主机上已经登录了你的账号，此时此刻 github 认为是该账号主人在操作这台主机，在配置 ssh 后就信任该主机了。所以下次在使用 git 的时候即使没有登录 github，也能直接从本地 push 代码到远程了。当然这里不要混淆了，你不能随意 push 你的代码到任何仓库，你只能 push 到你自己的仓库或者其他你有权限的仓库！

1.2.7.1 利用 Github CLI 新建远程仓库（Repositories）

- 安装 Github CLI
  gh 可以通过 Homebrew, MacPorts, Conda, Spack 等方式安装
  Homebrew

Install:	Upgrade:
`brew install gh`	`brew upgrade gh`

MacPorts

Install:	Upgrade:
`sudo port install gh`	`sudo port selfupdate && sudo port upgrade gh`

Conda

Install:	Upgrade:
`conda install gh --channel conda-forge`	`conda update gh --channel conda-forge`

Additional Conda installation options available on the gh-feedstock page.

Spack

Install:	Upgrade:
`spack install gh`	`spack uninstall gh && spack install gh`

- Git CLI 登陆验证
  - 生成 github token

    Personal access tokens > Generate new token > Generate new token (classic) > Note (What's this token for?) > Expiration > No Expiration > Select scopes > Select 'repo', 'read:org', 'admin:public_key'
  - 将上述生成的 token 复制保存以便后面使用
  - gh 登陆 github 验证

    `gh auth login`

```
alsichkann@Xins-iMac-2 MyWebsite % gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? SSH
? Upload your SSH public key to your GitHub account? /Users/alsichkann/.ssh/id_rsa.pub
? Title for your SSH key: GitHub CLI
? How would you like to authenticate GitHub CLI? Paste an authentication token
Tip: you can generate a Personal Access Token here https://github.com/settings/tokens
The minimum required scopes are 'repo', 'read:org', 'admin:public_key'.
? Paste your authentication token: ****************************************
- gh config set -h github.com git_protocol ssh
✓ Configured git protocol
HTTP 422: Validation Failed (https://api.github.com/user/keys)
key is already in use
```

- 新建 github repo 并将当前项目推送至远程 github repo

  `gh repo create`

```
alsichkann@Xins-iMac-2 Mac-Settings % gh repo create
? What would you like to do? Push an existing local repository to GitHub
? Path to local repository .
? Repository name macsoftset
? Description Notes of things that needed for work and research
? Visibility Public
✓ Created repository xinzhangseu/macsoftset on GitHub
? Add a remote? Yes
? What should the new remote be called? origin
✓ Added remote git@github.com:xinzhangseu/macsoftset.git
? Would you like to push commits from the current branch to "origin"? Yes
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 20 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (14/14), 235.87 KiB | 779.00 KiB/s, done.
Total 14 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:xinzhangseu/macsoftset.git
 * [new branch]      HEAD -> main
branch 'main' set up to track 'origin/main'.
✓ Pushed commits to git@github.com:xinzhangseu/macsoftset.git
```

- 后续本地仓库修改推送至远程 github repo

  `git push origin main`

1.3 利用 Git Action 发布 quarto 网页到 Netlify

GitHub Actions 是 GitHub 的持续集成服务。

Netlify 是静态网站自动部署平台，是目前比较简单的自动化部署方案。

部署静态网站到 Netlify 上有两种方法。

- 第一种是不用 GitHub actions，直接使用 netlify 连接到我们的 GitHub 仓库，netlify 会自动为我们设置好 CI/CD pipeline，只要我们 push 到相应的 branch 上，netlify 就会自动完成部署。我们也可以在 site setting 里对部署进行更多的设置。
- 第二种是不直接连接 Github repo 到 Netlify，用 GitHub Actions 设置 CI/CD workflow。

1.3.1 为什么用 Github Actions

首先，Netlify 的免费额度一个月只有 **300 分钟 build time**，如果我们有多个部署环境，经常部署的话，这点时间可能不够用。如果使用 GitHub Actions，会有 **2000 分钟的 build time**。其次，如果不使用 GitHub Actions，我们对 build process 没有什么选择，只能使用 Netlify 自带的设置，灵活性比较低。

1.3.2 使用 GitHub Actions 部署到 Netlify

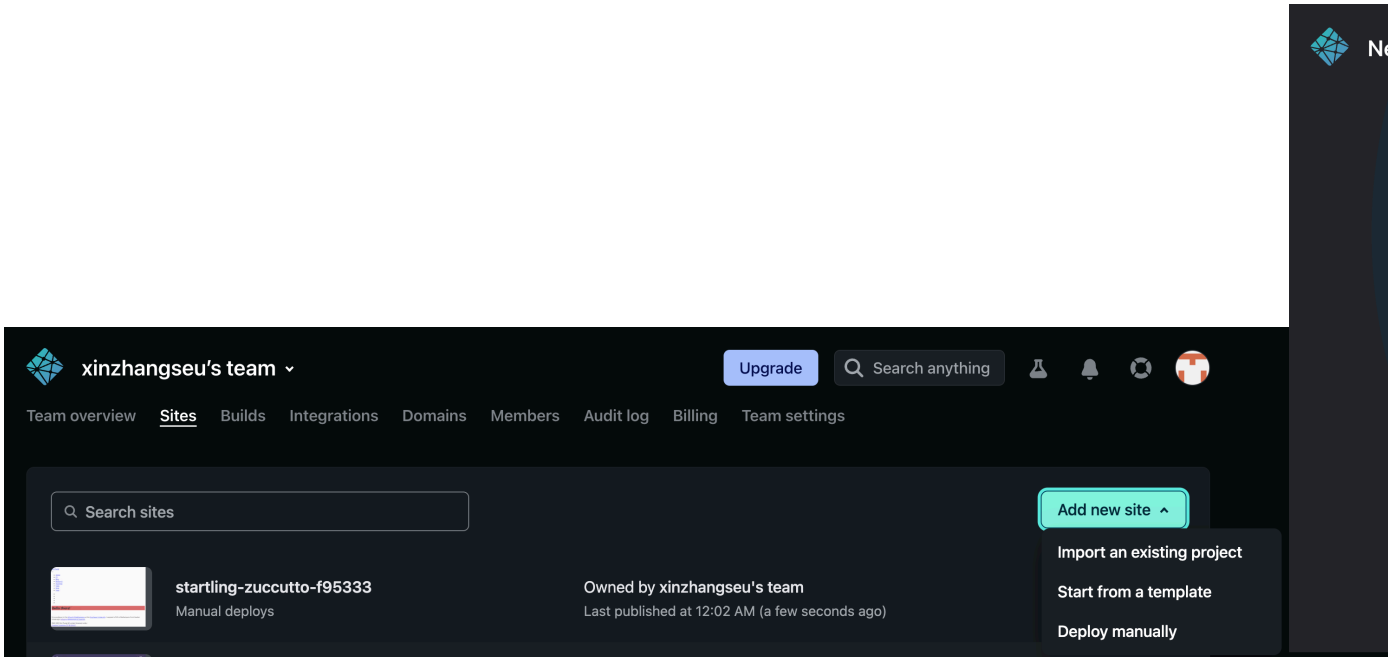
主要有 6 步：

- 创建 Netlify site
- 获取 `NETLIFY_AUTH_TOKEN` 和 `NETLIFY_SITE_ID`
- 设置 GitHub Repositories secrets
- 本地仓库设置 Quarto Publish
- 本地仓库设置 GitHub Actions

- 将本地仓库推送至 GitHub 远程仓库

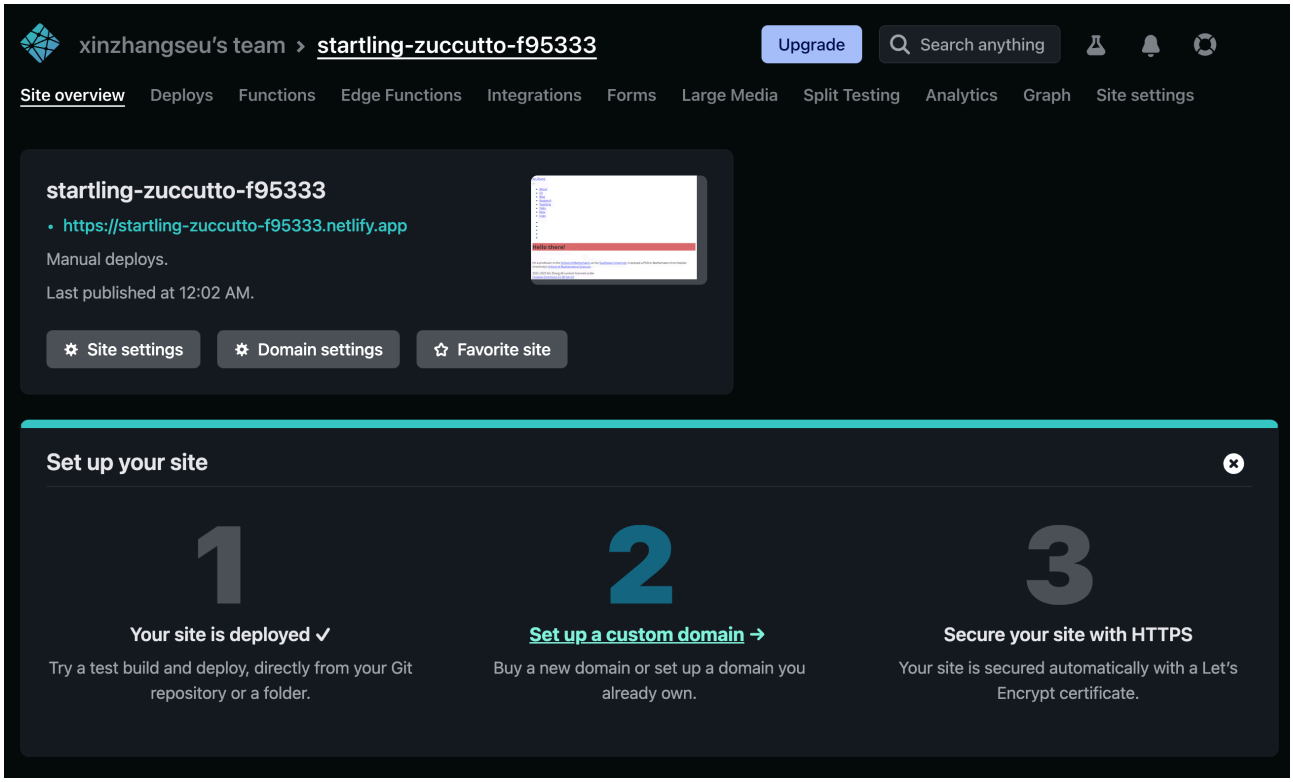
第一步

首先，登陆 Netlify > Sites > Add new site > Deploy manually



注意，这里不要点击 new site from Git，不要让 Netlify 直接连接到 GitHub。先拖拽一个空文件夹到上图虚线框里面，创建一个 Deploy manually Netlify Site，之后我们会通过 Github actions 上传我们的 build 到 Netlify。

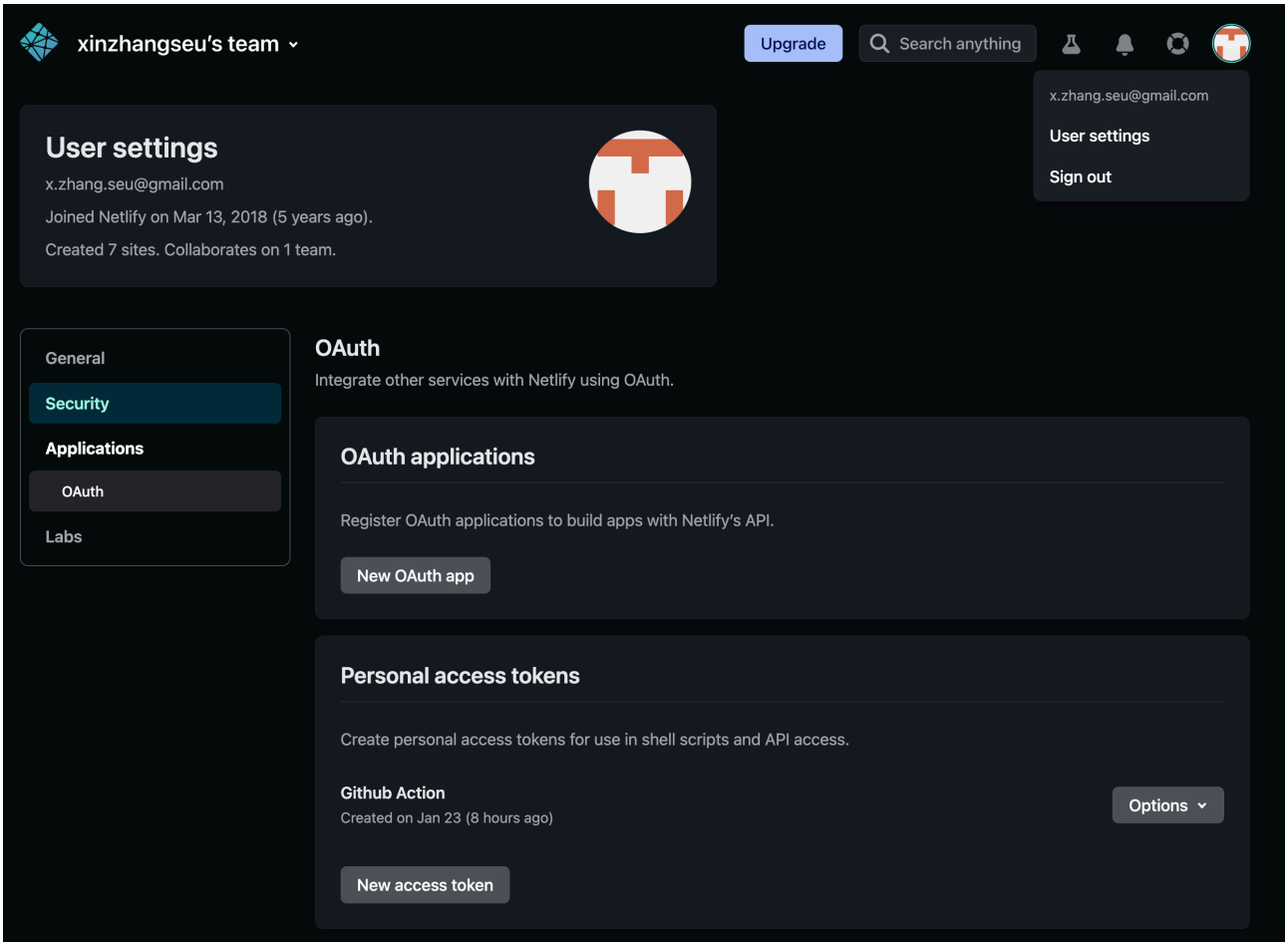
完成上传后，我么看到类似如下 Netlify Site 页面：



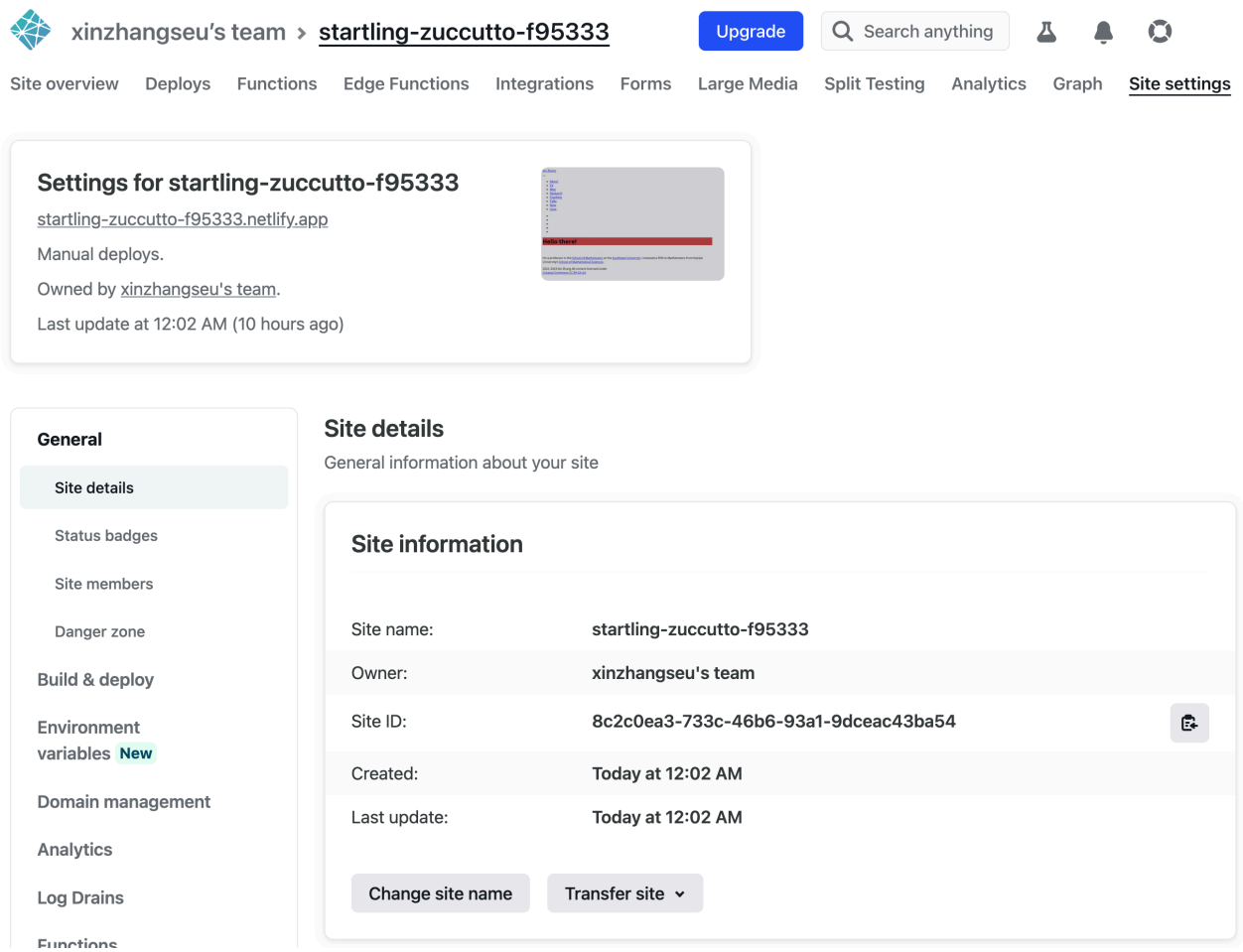
第二步

接下来，需要获取发布到 Netlify 所需的两个凭据并配置 GitHub Action.

第一个是 NETLIFY_AUTH_TOKEN，在 Netlify 页面右上角点击 User settings > Applications > Personal access tokens 中点击 New access token，创建之后先不要关闭这个页面，等下要复制粘贴。



第二个是 NETLIFY_SITE_ID，在第一步生成的 Neilify Site 页面最右侧上方点击 Site settings，然后可以看到在 Site Information 下面有一个 Site ID 就是我们需要的 NETLIFY_SITE_ID。

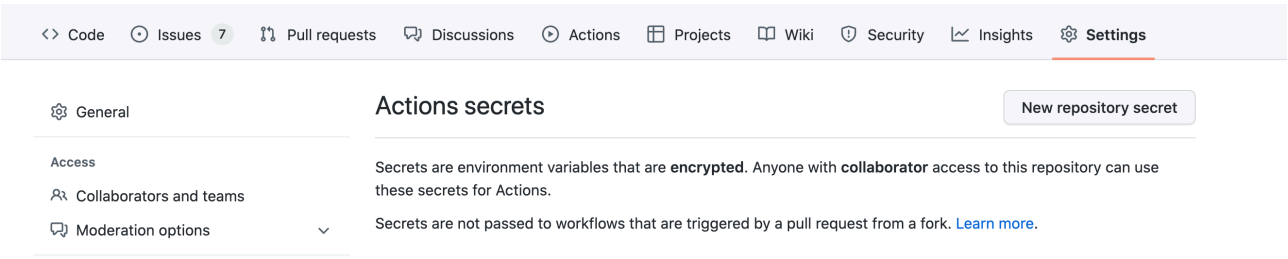


注意，系统自动生成的 Site name 较长，可以点击上图中的 Change site name 自行设置喜欢的 Site name。

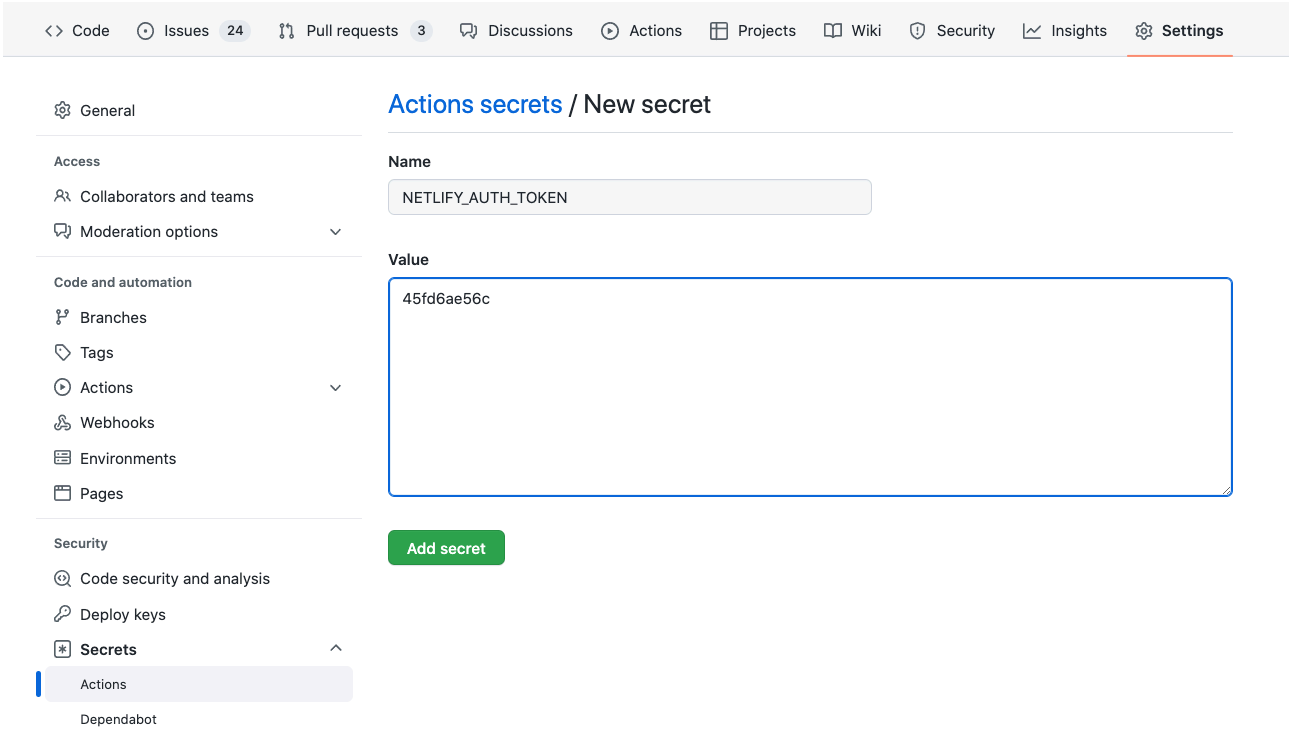
第三步

添加这两个 Netlify 凭据到 GitHub Repo 的 secrets 中。

在 GitHub 中点击准备发布到 Netlify 的仓库（repository）> Settings > Secrets and variables > Actions。



点击如上图所示的 New repository secret，在 Name 栏输入 NETLIFY_AUTH_TOKEN，Value 栏输入上一步生成的 NETLIFY_AUTH_TOKEN 值。



第四步

在本地项目仓库添加文件 _publish.yml，文件内容如下：

```
- source: project
  netlify:
    - id: "Netlify Site id"
      url: "https://Netlify Site name.netlify.app"
```

上述文件中的 “Netlify Site id”，“Netlify Site name” 可从第二步中的 Site information 中获取。

第五步

设置 GitHub Actions。在本地项目仓库新建文件夹.github/workflows/ 并在此文件夹中新建文件 publish.yml，其内容如下：

```
on:
  workflow_dispatch:
  schedule:
    # 每周四 早上 7 点
    - cron: '0 23 * * 3'
  push:
    branches: main

name: Quarto Publish to Netlify

jobs:
  build-deploy:
    runs-on: ubuntu-latest
```

```
env:
  GITHUB_PAT: ${{ secrets.GITHUB_TOKEN }}
  DISPLAY: ':99.0'
  RGL_USE_NULL: true
  LANG: "en_US.UTF-8"
  TZ: "Asia/Shanghai"
  ## RETICULATE_PYTHON_ENV: "/opt/.virtualenvs/r-tensorflow"
  ## CMDSTAN_VERSION: "2.29.2"
  ## CMDSTAN: "/opt/cmdstan/cmdstan-2.29.2"
  ## CMDSTANR_NO_VER_CHECK: true
steps:
  - name: Check out repository
    uses: actions/checkout@v3

  - name: Setup Pandoc
    uses: r-lib/actions/setup-pandoc@v2
    with:
      pandoc-version: '2.19.2'

  - name: Set up Quarto
    uses: quarto-dev/quarto-actions/setup@v2

  - uses: r-lib/actions/setup-r@v2
    with:
      use-public-rspm: true
      r-version: '4.2.2'
      extra-repositories: 'https://mc-stan.org/r-packages'

  - uses: r-lib/actions/setup-r-dependencies@v2
    with:
      packages:
        any::knitr
        any::rmarkdown
        any::downlit
        any::xml2

  - name: Render and Publish
    uses: quarto-dev/quarto-actions/publish@v2
    with:
      target: netlify
      NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
```

第六步

将本地仓库推送至 GitHub 远程仓库，从命令行进入 Git 项目目录，并执行

```
$ git add _publish_yml .github/
$ git commit -m "注释内容"
$ git push origin main
```

1.3.3 Quarto Books 设置

- _quarto.yml 文件内容

```
project:
  type: book
  output-dir: _book
  resources: elegantbook.cls
  ## 由于生成 pdf 需要 elegantbook 宏包，但该宏包不在 TexLive 仓库中。故将 elegantbook.cls 文件添加至项目并
lang: zh
book:
  title: " 书名"
  author: " 作者"
  date: "1/8/2023"
  chapters:
    - index.qmd
    - MacSettings.qmd
    - Routers.qmd
    - summary.qmd
    - references.qmd
  search: true
  downloads: [pdf]
  page-navigation: true
bibliography: references.bib
format:
  html:
    theme: cosmo
    crossref:
      chapters: true
      eq-prefix: ""
  pdf:
    documentclass: elegantbook
    classoption: [lang=cn,10pt]
    keep-tex: true
    include-in-header:
      text: |
        \usepackage{makeidx}
        \makeindex
    include-after-body:
      text: |
        \printindex
```

```
    colorlinks: true
    link-citations: true
    crossref:
      chapters: true
      eq-prefix: ""
editor: visual
```

- 安装 R 相依包的 DESCRIPTION 文件内容
  DESCRIPTION

```
Type: Book
Package: daar
Title: Mac Soft Setting.
Version: 0.01
Authors@R: c(
    person("Xin", "Zhang", , "x.zhang.seu@gmail.com", role = c("aut", "cre"))
  )
URL: https://https://macsoftset.netlify.app/,
     https://github.com/xinzhangseu/macsoftset/,
Depends:
    R (>= 4.2.0)
Imports:
    bayesplot (>= 1.10.0),
    beanplot,
    biscale (>= 1.0.0),
    broom,
    cartogram,
    cmdstanr (>= 0.5.3),
    data.table,
    datasauRus,
    downlit,
    dplyr,
    echarts4r,
    emo,
    emojifont,
    ganttrify,
    geodata (>= 0.4.13),
    geofacet,
    geomtextpath,
    gifski,
    ggalluvial,
    gganimate,
    ggbeeswarm,
    ggbump,
```

```
    ggChernoff,
    ggdensity,
    ggeffects,
    ggExtra,
    ggforce,
    ggmosaic,
    ggnewscale,
    ggplot2 (>= 3.4.0),
    ggradar,
    ggraph (>= 2.1.0),
    ggrepel,
    ggridges,
    ggsignif,
    ggstats,
    ggTimeSeries,
    ggVennDiagram,
    ggwordcloud,
    glmnet,
    gt (>= 0.8.0),
    HistData,
    keras,
    knitr,
    latex2exp,
    latticeExtra,
    likert,
    lme4,
    lvplot (>= 0.2.1),
    magick,
    pals,
    patchwork,
    pdftools,
    plot3D,
    plotly (>= 4.10.1),
    purrr (>= 1.0.0),
    quarto,
    ragg,
    reticulate,
    rgl,
    rootSolve,
    scatterplot3d,
    sf (>= 1.0.9),
    showtext,
    spatstat.explore (>= 3.0.0),
    spatstat.geom (>= 3.0.0),
```

```
    stars (>= 0.6.0),
    statebins,
    tensorflow,
    terra (>= 1.6-41),
    tidycensus,
    tidygraph,
    tidyterra (>= 0.3.1),
    titanic,
    tikzDevice,
    treemapify,
    vcd,
    vioplot,
    webshot2,
    xml2
Remotes:
    davidsjoberg/ggbump,
    giocomai/ganttrify,
    hadley/emo,
    corybrunson/ggalluvial,
    ricardo-bion/ggradar,
    jbryer/likert
Suggests:
    rsconnect
Additional_repositories: https://mc-stan.org/r-packages/
Encoding: UTF-8
License: CC NC ND 4.0
```

- TexLive 宏包文件：texlive.txt

```
animate
appendixnumberbeamer
arev
awesomebox
bookmark
cancel
caption
carlito
cm-super
colortbl
ctex
datatool
dejavu
doublestroke
draftwatermark
```

```
dvipng
ebgaramond
ebgaramond-maths
environ
epstopdf-pkg
etex-pkg
everypage
everysel
fancyhdr
filehook
fira
fontaxes
fontawesome5
grfext
helvetic
hyphen-german
iitem
jknapltx
koma-script
listings
lm-math
makeindex
marginfix
mathdesign
mdwtools
media9
metalogo
microtype
ms
multirow
ncntrsbk
needspace
newtx
nimbus15
oberdiek
ocgx2
oldstandard
opensans
pgf
pgfornament
pgfornament-han
pgfopts
pgfplots
preview
```

```
psnfss
ragged2e
realscripts
relsize
rsfs
savesym
setspace
smartdiagram
soul
sourcecodepro
sourcesanspro
sourceserifpro
standalone
subfig
tabu
tcolorbox
textpos
threeparttable
threeparttablex
tikz-network
titlepic
titlesec
tocbibind
tocloft
trimspaces
tufte-latex
type1cm
ucs
ulem
unicode-math
varwidth
xcolor
xecjk
xltxtra
xpatch
xstring
zhnumber
zref
```

- GitHub Action 配置

```
on:
  workflow_dispatch:
  schedule:
```

```
    # 每周四 早上 7 点
    - cron: '0 23 * * 3'
  #push:
    #branches: main
  #pull_request:
    #branches: main
name: Quarto Publish to Netlify
env:
  isExtPR: ${{ github.event.pull_request.head.repo.fork == true }}
  RUST_BACKTRACE: 1

jobs:
  build-deploy:
    if: "!contains(github.event.head_commit.message, '[docker]')"
    runs-on: ubuntu-22.04
    env:
      GITHUB_PAT: ${{ secrets.GITHUB_TOKEN }}
      DISPLAY: ':99.0'
      RGL_USE_NULL: true
      LANG: "en_US.UTF-8"
      TZ: "Asia/Shanghai"
      RETICULATE_PYTHON_ENV: "/opt/.virtualenvs/r-tensorflow"
      CMDSTAN_VERSION: "2.29.2"
      CMDSTAN: "/opt/cmdstan/cmdstan-2.29.2"
      CMDSTANR_NO_VER_CHECK: true
    steps:
      - uses: actions/checkout@v3
      - uses: r-lib/actions/setup-pandoc@v2
        with:
          pandoc-version: '2.19.2'

      - name: Install Quarto
        uses: quarto-dev/quarto-actions/setup@v2
        with:
          version: 1.2.280
      - run: |
          quarto --version
          quarto pandoc --version
          pandoc --version

      - name: Install TinyTeX
        uses: r-lib/actions/setup-tinytex@v2
        env:
          # install full prebuilt version
```

```
    TINYTEX_INSTALLER: TinyTeX

- uses: r-lib/actions/setup-r@v2
  with:
    use-public-rspm: true
    r-version: '4.2.2'
    extra-repositories: 'https://mc-stan.org/r-packages'

# based on the renv lockfile file
# - uses: r-lib/actions/setup-renv@v2
# based on the DESCRIPTION file
- uses: r-lib/actions/setup-r-dependencies@v2
  with:
    needs: book

- name: Install Fonts From System
  run: |
    sudo apt-get install -y fonts-noto-core fonts-noto-cjk fonts-noto-color-emoji
    fc-list | grep 'noto' | sort
    fc-list | sort
    fc-cache -fsv

- name: Install LaTeX packages
  run: |
    if(!require('tinytex')) install.packages('tinytex')
    tinytex::tlmgr_install(readLines("texlive.txt"))
  shell: Rscript {0}

- name: Check TikZ
  run: |
    if(!require('tikzDevice')) install.packages('tikzDevice')
    tikzDevice::tikzTest()
  shell: Rscript {0}

- name: Check R and xvfb
  run: |
    xvfb-run Rscript -e 'capabilities()'

- name: Render book to all format
  # Add any command line argument needed
  run: |
    make all

- name: Render and Publish
```

```
        uses: quarto-dev/quarto-actions/publish@v2
        with:
          target: netlify
          NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
```

1.3.4 Mac Port install log

```
To use this bootstrap version of cmake instead of the usual cmake port, add
    the following lines to the Portfile:

    depends_build-replace  path:bin/cmake:cmake port:cmake-bootstrap
    configure.cmd          ${prefix}/libexec/cmake-bootstrap/bin/cmake
  gawk has the following notes:
    readline support has been removed from gawk. If you need to run gawk
    interactively, install rlwrap:

    sudo port install rlwrap

    and run gawk using rlwrap:

    rlwrap gawk ...
  libomp has the following notes:
    To use this OpenMP library:
     * For clang-3.8+, or clang-3.7 with +openmp variant:
       add "-fopenmp" during compilation / linking.
     * For clang-3.7 without +openmp variant, use:
       "-I/opt/local/include/libomp -L/opt/local/lib/libomp -fopenmp"
  libpsl has the following notes:
    libpsl API documentation is provided by the port 'libpsl-docs'.
  py310-cython has the following notes:
    To make the Python 3.10 version of Cython the one that is run when you
    execute the commands without a version suffix, e.g. 'cython', run:

    port select --set cython cython310
  py310-pygments has the following notes:
    To make the Python 3.10 version of Pygments the one that is run when you
    execute the commands without a version suffix, e.g. 'pygmentize', run:

    port select --set pygments py310-pygments
  python310 has the following notes:
    To make this the default Python or Python 3 (i.e., the version run by the
    'python' or 'python3' commands), run one or both of:

        sudo port select --set python python310
```

```
      sudo port select --set python3 python310
python311 has the following notes:
  To make this the default Python or Python 3 (i.e., the version run by the
  'python' or 'python3' commands), run one or both of:

      sudo port select --set python python311
      sudo port select --set python3 python311
```

第 2 章 Git 和 github

git 是版本控制系统, 而 github 是开源代码托管平台, 提供的是基于 git 的开源代码托管服务. 对于一个团队来说, 即使没有 github, 也可以通过自己搭建 git 服务器来进行代码的管理, 甚至还有一些其他的基于 git 的代码托管平台使用, 比如 gitlab,gitee(码云) 等.

2.1 安装 git

Mac 和 Linux 是默认安装 git 的, 打开 terminal, 输入 `git version`, 如果输出 git 版本号, 则说明已有 git.

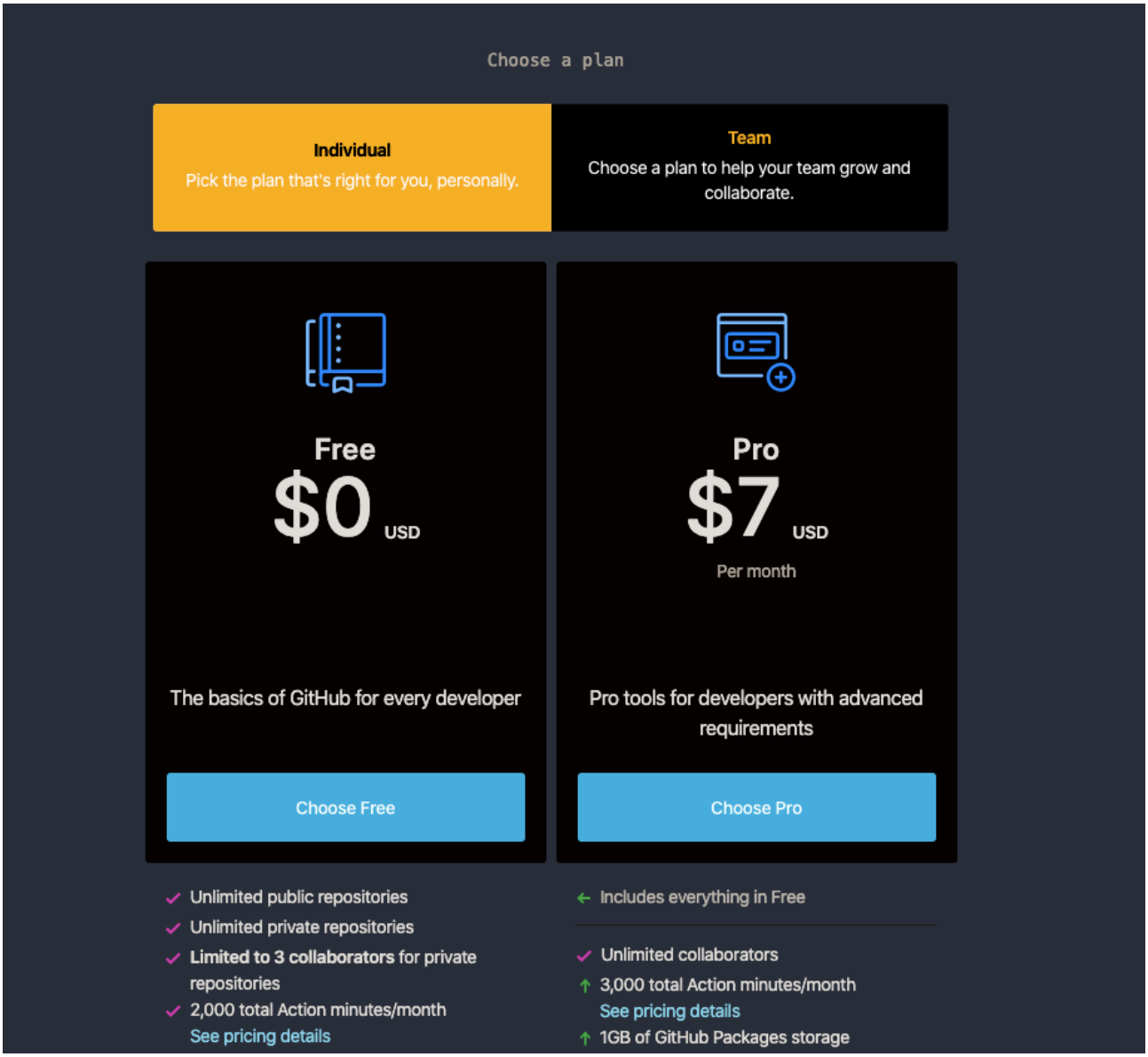
windows 安装 git 可以直接到官网下载, 然后安装即可. 官网如下:

https://git-scm.com/downloads

安装结束之后, 在开始菜单如果有 `git bash`, 则说明安装成功.

2.2 注册 github 账户

1. 打开 github 官网注册界面
   https://github.com/join?source=header-home
2. 填入自己的账户名, 邮箱和密码.
   选择 `free` plan 即可.



3. 选择或者填写一些内容.
4. 验证邮箱

2.3 Git 基础知识

`git` 是 Linux 的作者写的. 是一个分布式的版本控制软件.

主要功能:

- 备份代码.
- 版本管理.
- 协作办公.

git 的基本概念:

工作区 (work directory): 简单来说就是你的电脑里的目录, 也就是某个本地文件夹.

仓库 (Respository): 工作区有一个隐藏目录`.git`, 这个不算是工作区, 而是 git 的仓库, 也称之为版本库, 这

里存储的就是所有代码的版本.

暂存区 (stage 或者 index):git 的仓库中存放了很多东西, 其中最为重要的就是暂存区.

远程仓库 (remote directory): 本地仓库的东西如果需要托管到 github 上, 就需要在 github 上创建一个仓库 (远程仓库), 然后将本地仓库和远程仓库联系起来, 这样就可以将本地仓库的内容推送到远程仓库.

2.4 Git 配制

2.4.1 安装 git 之后, 需要进行一些全局设置, 比如用户名邮箱.

主要, 下面的所有操作, 命令,max 和 linux' 是在终端 (terminal) 中进行的, 在 windows 中, 是在 git bash 中.

设置的主要命令是 git config:

```
git config --global user.name "your name"
git config --global user.email "your email"
```

其中,--global 是指全局配置, 如果不写 (或者写为 local), 则只为当前仓库 (repository) 配置.

如果有多个 git 用户, 这时候需要先确定某个仓库使用的哪个账户.

```
git config user.name
git config user.email
```

查询全局的 git 账户信息:

```
git config --global user.name
git config --global user.email
```

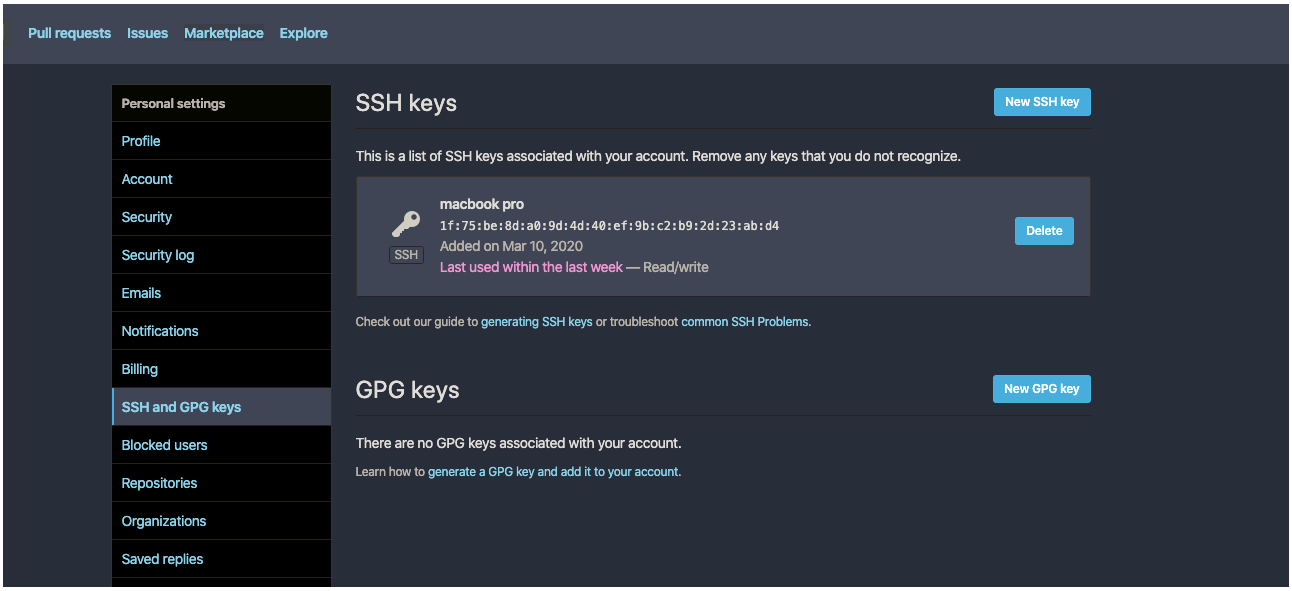
如果想要去掉全局用户和邮箱设置.

```
git config --global --unset user_name
git config --global --unset user_email
```

2.4.2 生成秘钥

```
ssh-keygen -t rsa -C "shenxt1990@163.com"
```

然后会在 ~/.ssh/文件夹中会产生秘钥文件, 我们只需要将公钥 (pub 后缀) 的内容拷贝到 github 中即可.



对于 windows 用户, 请参考这个帖子:

https://www.jianshu.com/p/95262f5eba7a

2.5 Git 基本操作

2.5.1 创建本地 git 仓库 (reop)

使用 `git init` 命令

首先, 创建一个你要当作仓库的文件夹, 然后将其设置为工作路径, 然后执行.

```
mkdir git_test# 创建本地文件夹
cd git_test# 进入到该文件夹将其作为工作目录
git init# 创建本地 git 仓库
```

这时候查看目录结构, 就可以看到创建了一个名为`.git` 的子目录, 这就说明创建版本库成功了.

```
ls -la
```

2.5.2 将文件添加到版本库

要将一个文件纳入到版本管理, 首先需要将其添加到暂存区 (stage), 然后才能提交到仓库中.

- 使用 `git add` 命令将文件添加到暂存区:

比如我们首先新建一个名为 `README.md` 的文件, 然后将其添加到暂存区.

```
touch README.md# 创建文件
git add README.md# 将该文件加入到暂存区
```

当然也可以使用下面命令将所有修改添加到暂存区:

```
git add .
```

. 是正则表达式, 代表任意文件. 但是空文件夹是不会添加到暂存区的.

- 使用 git commit 命令将暂存区文件提交到仓库中.

```
git commit # 如果暂存区有文件, 则将其中的文件提交到仓库
git commit -m "your note for this commit" # 用于注明提交的内容, 变更等信息, 方便溯源
```

如果直接使用 git commit 提交, 不带注释信息, 则会先弹出评论界面, 需要评论.

注意这时候是使用 vi 打开的文件, 所以需要掌握一点 vi 的基础知识. 参考下面帖子:

https://www.cnblogs.com/itech/archive/2009/04/17/1438439.html

简单来说, 首先在命令行模式下按下字母 i 进入插入模式, 在该模式下进行信息的输入. 输入结束之后, 点击 ESC 退回到命令行模式, 然后再输入冒号 (:). 输入 wq, 保存文件修改退出 vi 编辑器.

2.5.3 查看仓库状态

不论我们是新建了文件, 将文件加入暂存区, 或者是其他修改等, 我们都可以使用 git status 来查看当前的仓库状态.

如果当前没有任何变动.

```
WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git status
On branch master
nothing to commit, working tree clean
```

如果我们新建了一个文件, 然后再次查看状态:

因为我们还没有将其添加到暂存区, 因此显示该新建文件为 Untracted files

```
WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ touch test.txt

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.txt

nothing added to commit but untracked files present (use "git add" to track)

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$
```

将文件放入暂存区, 然后再查看状态:

```
WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git add .

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   test.txt


WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$
```

然后再将其进行提交, 查看状态.

```
WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git commit
[master d058b56] creat a new file test.txt.
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.txt

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git status
On branch master
nothing to commit, working tree clean

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$
```

2.5.4 查看仓库中的具体修改

如果做过之后修改, 我们忘了做了哪些修改, 可以使用 `git diff` 命令来查看具体修改内容.

```
git diff # 查看所有改动
git diff README.md # 查看具体文件的改动
```

比如我们在 `test.txt` 文件中加入一行, 然后查看:

```
git diff test.txt
```

```
WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ vi test.txt

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git diff
warning: LF will be replaced by CRLF in test.txt.
The file will have its original line endings in your working directory
diff --git a/test.txt b/test.txt
index e69de29..c90251e 100644
--- a/test.txt
+++ b/test.txt
@@ -0,0 +1 @@
+add new line in test.txt

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
```

可以看到记录了对该文件的修改内容.

注意, 如果你修改之后已经将其存入暂存区, 则 `git diff` 不会再显示内容.

```
+add new line in test.txt

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git add .
warning: LF will be replaced by CRLF in test.txt.
The file will have its original line endings in your working directory

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git diff

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$
```

2.5.5 查看提交的历史记录

有的时候, 你需要查看自己做过哪些提交 (commit), 来回顾自己的完成部分, 这时候需要使用 `git log` 命令.

```
git log # 显示所有提交的历史记录
git log --pretty=oneline # 单行显示提交历史记录的内容
```

```
$ git log
commit ec5d68882b35a23348fdf6cb887b4dded5031158 (HEAD -> master)
Author: Xiaotao Shen <shenxt1990@163.com>
Date:   Wed Mar 4 10:58:30 2020 -0800

    add a new line in test.txt

commit d058b567e22d0ab4ebae40b386bf0dca4cb6ec8b
Author: Xiaotao Shen <shenxt1990@163.com>
Date:   Wed Mar 4 10:45:53 2020 -0800

    creat a new file test.txt.

commit 6fc24b1b4acfa00c4f671938357d6ee1a80d8b55
Author: Xiaotao Shen <shenxt1990@163.com>
Date:   Wed Mar 4 10:25:30 2020 -0800

    add a line in README.md

commit 71c4c510c759d03b039412f3f396b3cb8804f92c
Author: Xiaotao Shen <shenxt1990@163.com>
Date:   Wed Mar 4 10:23:27 2020 -0800

    creat readme.md

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$
```

可以看到我们现在一共进行了 4 次提交, 每次提交的第一行 commit 之后是该次提交的唯一 ID. 然后后面显示提交者, 时间以及一些记录等.

如果提交非常多, 需要显示内容精简一些, 就需要使用 `git log --pretty=oneline` 命令.

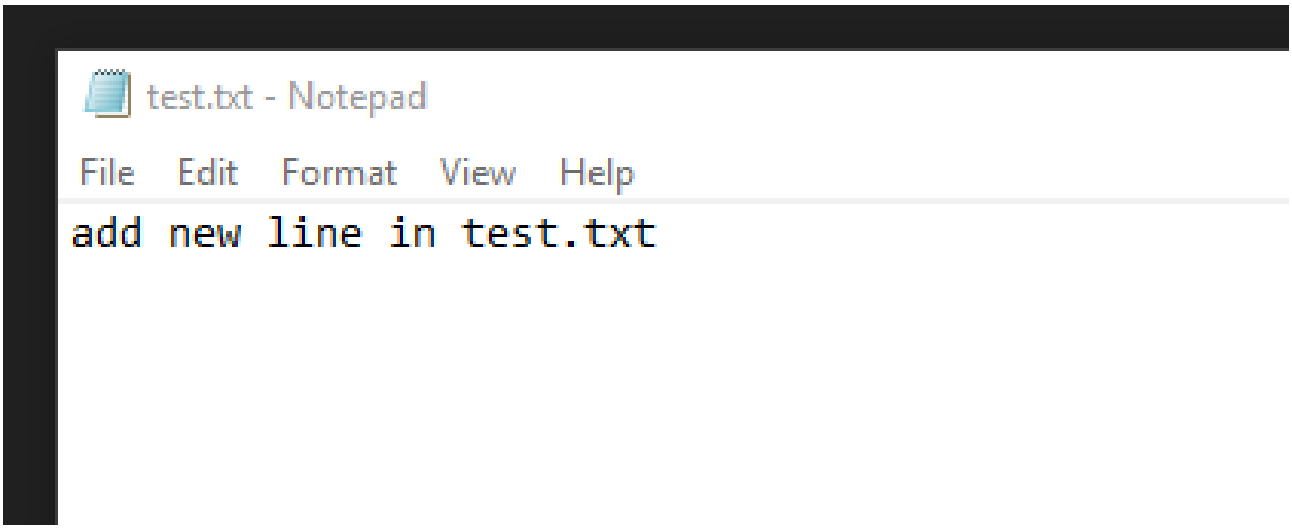
```
WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git log --pretty=oneline
ec5d68882b35a23348fdf6cb887b4dded5031158 (HEAD -> master) add a new line in test.txt
d058b567e22d0ab4ebae40b386bf0dca4cb6ec8b creat a new file test.txt.
6fc24b1b4acfa00c4f671938357d6ee1a80d8b55 add a line in README.md
71c4c510c759d03b039412f3f396b3cb8804f92c creat readme.md

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$
```

2.5.6 版本回退

有了 `git log` 来查看提交的历史记录, 我们就可以通过 `git reset --hard` 来退回到我们需要的某个特定版本.

```
git reset --hard HEAD^ # 回退到上一个提交版本
git reset --hard HEAD^^ # 回退到上上一个提交版本
git reset --hard 'commit_id' # 会退到 commit_id 指定的提交版本
```

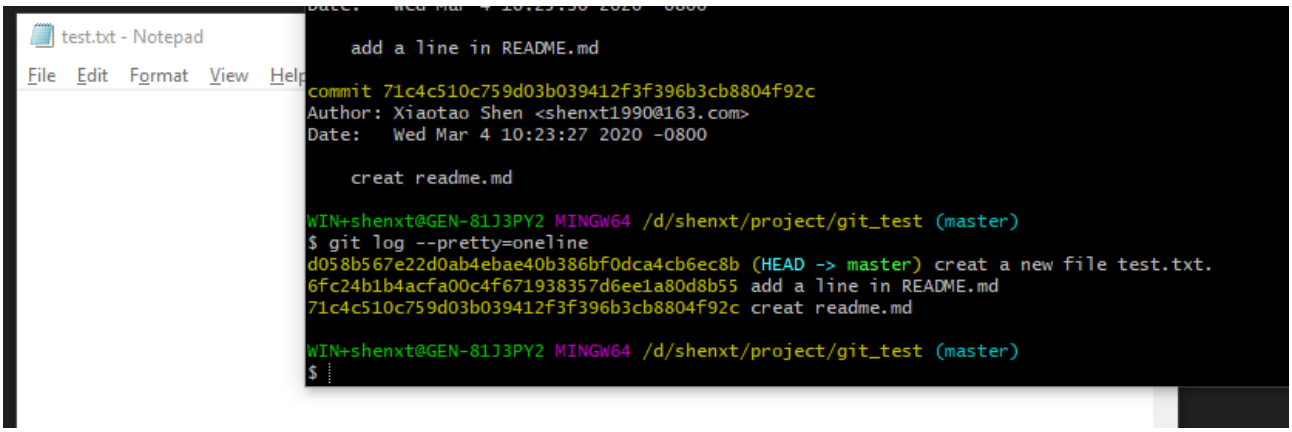
使用 `git log` 我们可以看到每个提交都有一个唯一的 commit ID, 其中上一个也可以使用 `HEAD` 代表. 因此我们可以退回到上一个版本. 比如, 我们在最近的一次提交中修改了 test.txt, 在其中加入一行.



我们想退回到上一个版本, 不想加入这一行, 可以使用下面命令.

```
git reset --hard HEAD^
```

这时候再查看 log, 可以看到最近的一个 commit 已经消失, 回到了上一个提交. 而上一次的修改也已经消失.

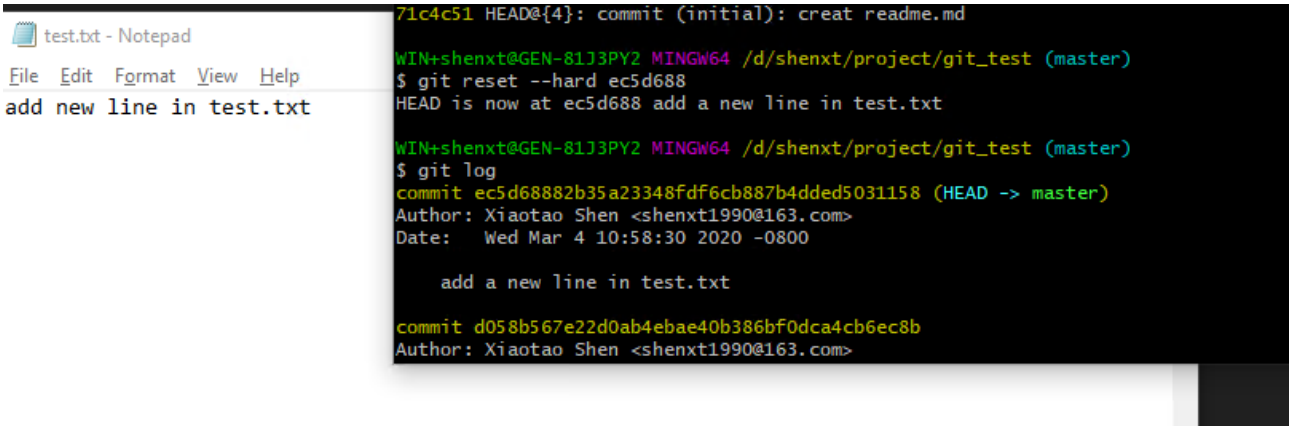


2.5.7 回到未来某个版本

退回到原来某个版本之后, 如何再回到未来某个版本呢? 还是使用 `git reset --hard` 命令, 但是此时使用 `git log` 命令是无法显示在这之后的提交信息的。但是，通过 `git reflog` 可以获取到操作命令的历史。

```
WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git log --pretty=oneline
d058b567e22d0ab4ebae40b386bf0dca4cb6ec8b (HEAD -> master) creat a new file test.txt.
6fc24b1b4acfa00c4f671938357d6ee1a80d8b55 add a line in README.md
71c4c510c759d03b039412f3f396b3cb8804f92c creat readme.md

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git reflog
d058b56 (HEAD -> master) HEAD@{0}: reset: moving to HEAD^
ec5d688 HEAD@{1}: commit: add a new line in test.txt
d058b56 (HEAD -> master) HEAD@{2}: commit: creat a new file test.txt.
6fc24b1 HEAD@{3}: commit: add a line in README.md
71c4c51 HEAD@{4}: commit (initial): creat readme.md

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$
```

这时候只要在使用 `git reset` 命令就可以回到未来的任意一个版本了. 而且这时候再使用 `git log` 也回到了最初的版本提交记录.



2.5.8 撤销修改

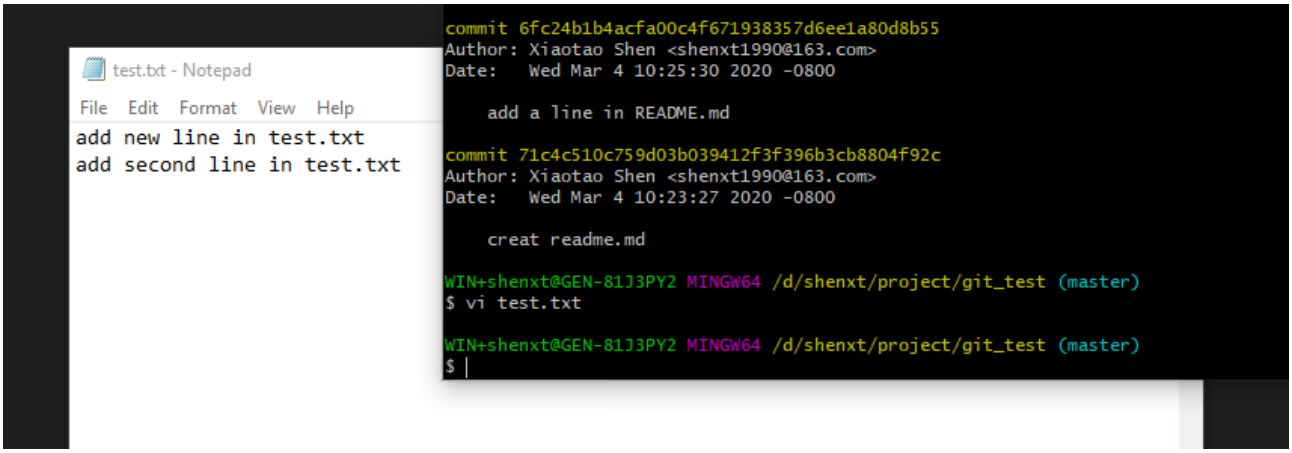
撤销修改同样包括两方面的内容，由于仓库中的文件在提交 (commit) 之前，可能在工作区中 (也就是还未运行 `git add` 命令)，尚未在版本控制范围内，也可能在暂存区中 (也就是运行了 `git add` 命令但是没有运行 `git commit` 命令)。

2.5.8.1 丢弃工作区中的文件修改

使用如下命令.

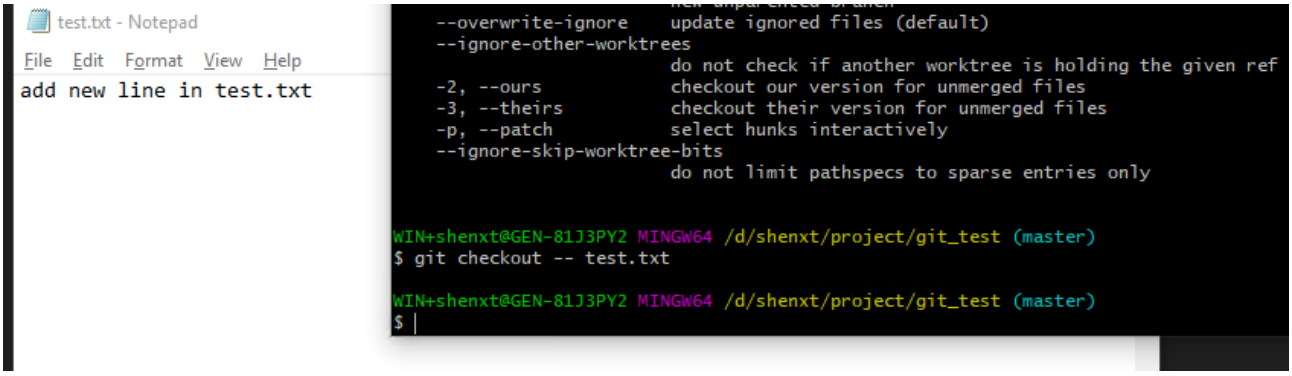
```
git checkout -- README.md     # 如果 README.md 文件在工作区，则丢弃其修改
git checkout -- .             # 丢弃当前目录下所有工作区中文件的修改
```

比如我们再 `test.txt` 中添加另外一行, 然后不加入暂存区, 然后将其丢弃掉.



然后运行:

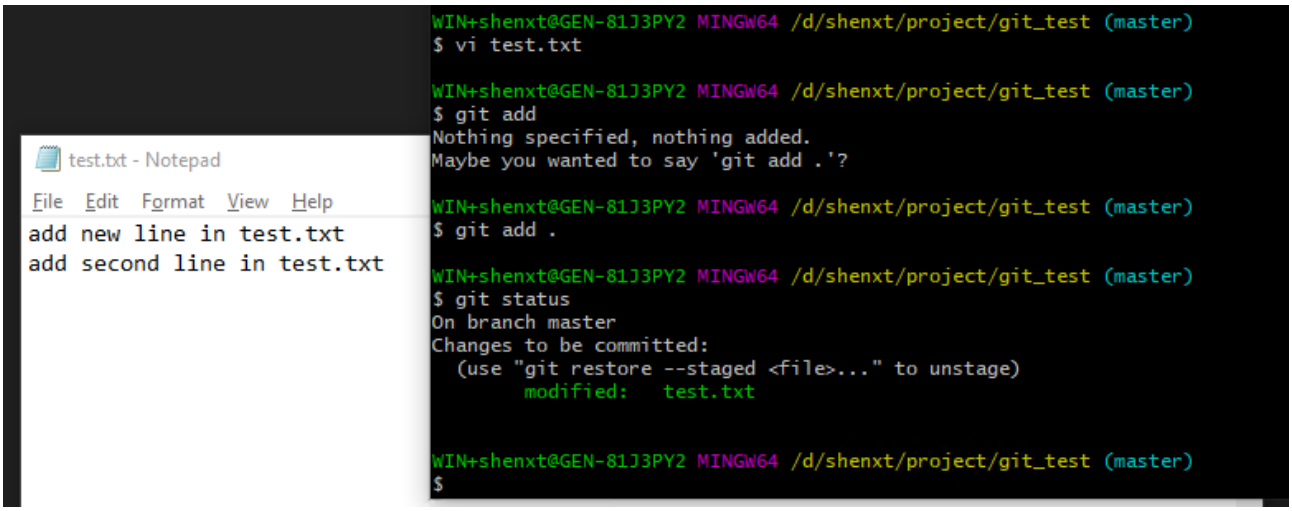
```
git checkout -- test.txt
```



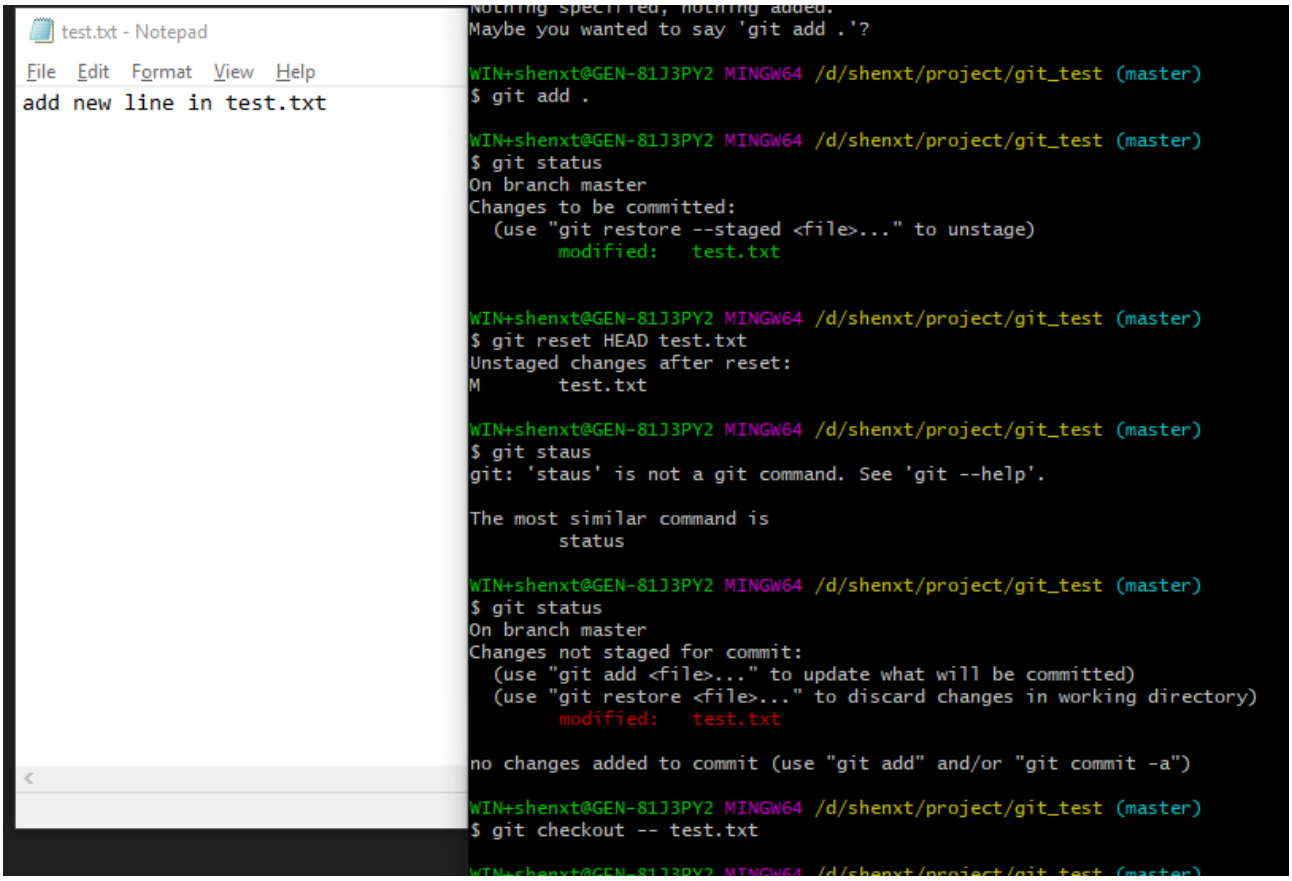
2.5.8.2 丢弃暂存区的修改

如果已经运行了 `git add` 将修改放入了暂存区, 可以通过下列命令进行丢弃:

```
git reset HEAD README.md # 将 README.md 恢复到 HEAD 提交版本的状态
git checkout -- README.md
```



然后将其删除掉.



2.5.9 删除文件

在文件未添加到暂存区之前, 对想删除文件可以直接物理删除. 或者通过 `git checkout -- file` 来丢弃. 如果文件已经被提交, 则需要 `git rm` 来删除.

```
git rm Readme.md # 删除已经提交 (commit) 过的文件
```

注意: `git rm` 只能删除已经提交到版本库中的文件. 其他状态的文件直接用这个命令操作是出错的.

```
WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git status
On branch master
nothing to commit, working tree clean

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ git rm test.txt
rm 'test.txt'

WIN+shenxt@GEN-81J3PY2 MINGW64 /d/shenxt/project/git_test (master)
$ ls -la
total 9
drwxr-xr-x 1 WIN+shenxt 2147484161 0 Mar  4 20:40 ./
drwxr-xr-x 1 WIN+shenxt 2147484161 0 Mar  4 10:07 ../
drwxr-xr-x 1 WIN+shenxt 2147484161 0 Mar  4 20:40 .git/
-rw-r--r-- 1 WIN+shenxt 2147484161 4 Mar  4 10:24 README.md
```

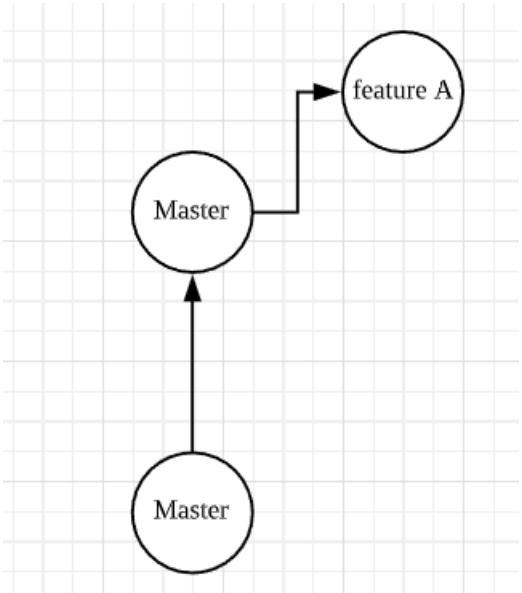
2.6 git 的分支管理

分支管理是版本管理中的重要概念. 在进行多个并行作业时, 我们经常会用到分支. 在这类并行开发的过程中, 往往同时存在着多个最新的代码状态.

master 分支时 git 默认创建的分支. 因此基本上所有的开发都是以这个分支为中心进行的.

在不同分支中, 可以同时进行不同的作业, 等该分支的作业完成之后, 再与 master 分支进行合并.

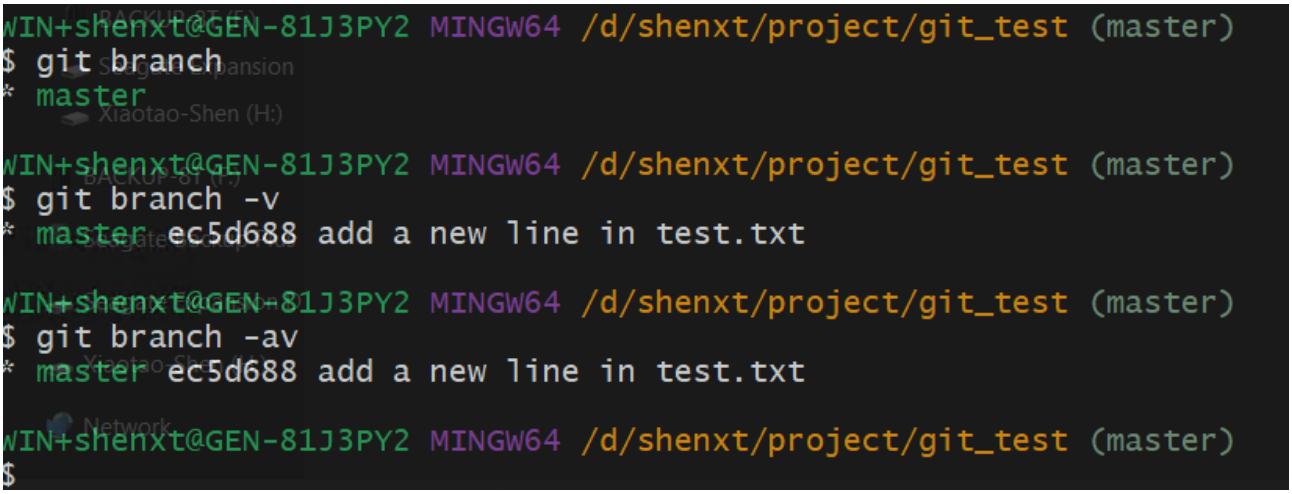
比如下图中, 我们首先以 master 作为模板复制得到 `feature A` 的分支.`feature A` 分支主要用来开发另外一个新的特性, 等开发完毕之后, 再将其合并到 master 分支中即可.



2.6.1 查看分支

使用 `git branch` 命令查看分支信息.

```
git branch        # 查看本地分支信息
git branch -v     # 查看相对详细的本地分支信息
git branch -av     # 查看包括远程仓库在内的分支信息
```



可以看到我们现在只有一个分支, 也就是 master, 然后星号标识的就是当前所在的分支.

2.6.2 创建分支

使用 `git branch new.name` 用来创建新的分支.

```
git branch development ## 创建一个新的名字叫做 development 的分支
```

需要注意的是, 在哪个分支下面新建分支, 则该分支就是 copy 自当前所在的分支.

2.6.3 切换分支

使用 `git checkout` 命令来切换分支. 比如我们先新建一个 development 的分支, 然后切换到该分支下.

```
git branch development
git checkout development
```

当然也可以使用下面命令直接创建并切换到新建的分支下.

```
git checkout -b development
```

2.6.4 switch 命令

我们注意到切换分支使用 `git checkout <branch>`, 而前面讲过的撤销修改则是 `git checkout -- <file>`, 同一个命令, 有两种作用, 确实有点令人迷惑.

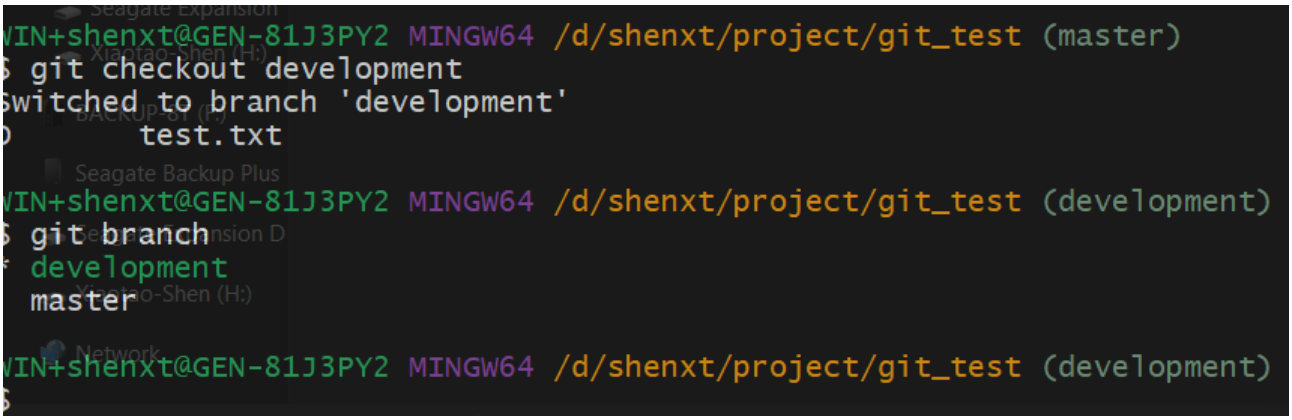
实际上, 切换分支这个动作, 用 `switch` 更科学。因此, 最新版本的 Git 提供了新的 `git switch` 命令来切换分支:

创建并切换到新的 development 分支, 可以使用:

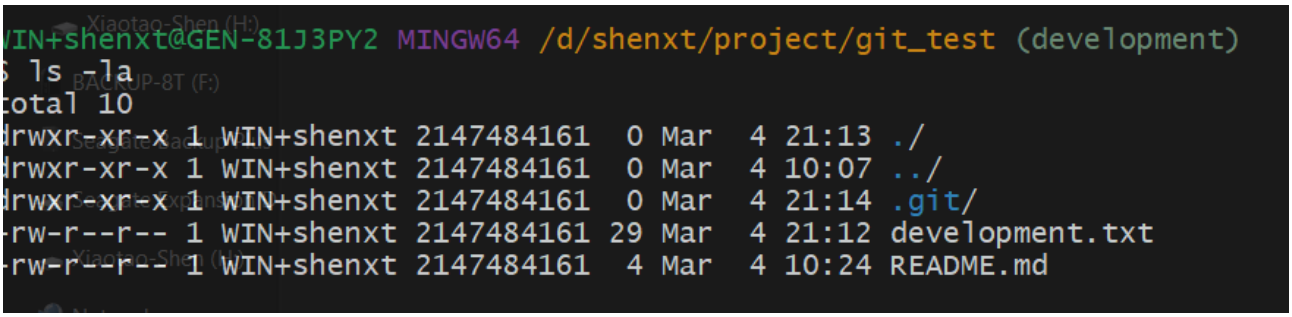
```
git switch -c development
```

直接切换到已有分支:

```
git switch master
```



我们下面在 development 分支下面再新建一个 `development.txt` 文件, 然后写入一些内容.



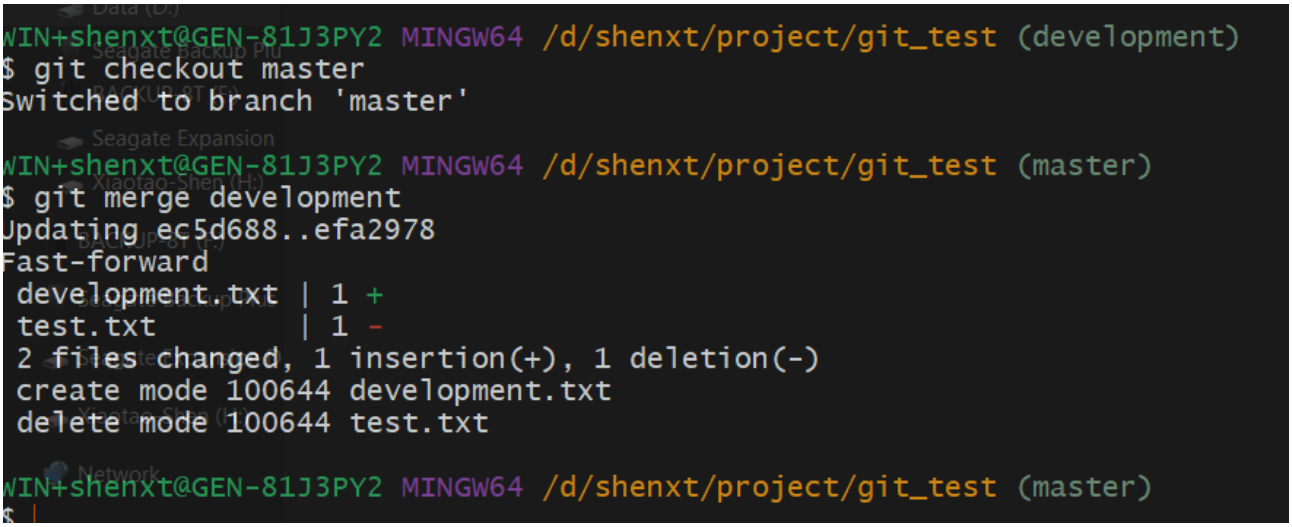
可以看到此时有一个新建的 development.txt 文件. 如果我们切换到 master 分支, 还是有该文件, 但是对该文件内容进行修改, 则因为没有加入到暂存区, 因此并不会进行版本控制.

2.6.5 合并分支 (merge)

当我们修复完成一个 Bug, 或者开发完成一个新特性, 我们就会把相关的 Bug 或者特性的上修改合并回原来的主分支上, 这时候就需要 `git merge` 命令来做分支的合并.

首先需要切换回最终需要合并的分支, 也就是 `master` 分支, 然后再合并.

```
git checkout master   # 切换回 master 分支
git merge development # 将 development 分支中的修改合并回 master 分支
```



2.6.6 删除分支

当之前创建的分支, 完成了它的使命, 如 Bug 修复完, 分支合并以后, 这个分支就不在需要了, 就可以删除它.

```
git branch -d development # 删除 development 分支
```

如果分支没有合并到 master 分支中, 直接使用上面命令会报错, 可以使用下面的命令强制删除.

```
git branch -D development
```

2.6.7 分支提交冲突

如果新创建了一个分支 `develpment`, 然后在该分支下对 `readme.txt` 进行修改, 然后提交.

随后然后切换到 `master` 分支, 然后也对 `readme.txt` 进行修改, 然后也提交.

这样, 两个分支分别有了不同的提交. 这种情况下, 再进行合并就会有冲突.

我们合并一下试试:

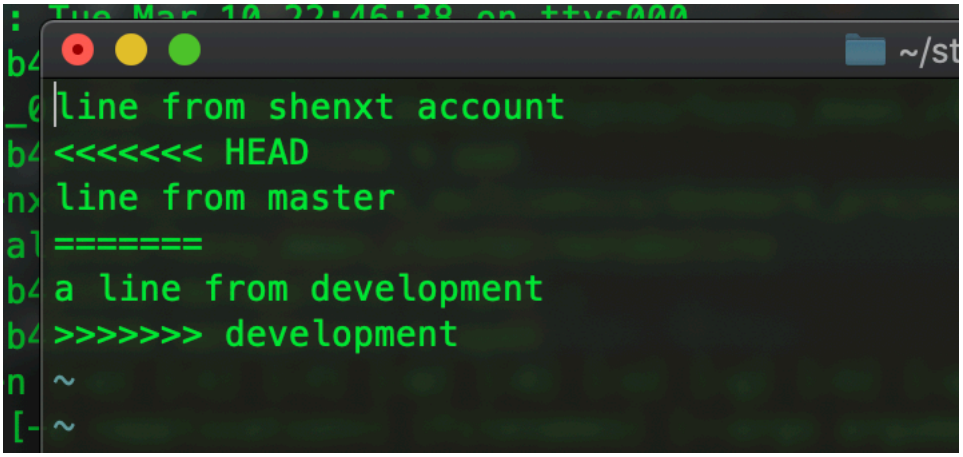
```
![](images/Screen Shot 2020-03-11 at 9.20.30 PM.png)
```

可以看到报错,Git 告诉我们,`readme.txt` 文件存在冲突, 必须手动解决冲突后再提交.`git status` 也可以告诉我们冲突的文件:

```
git status
```

```
Automatic merge failed; fix conflicts and then commit the result.
[shenxt@GEN-C02C93JZMD6R github_test3 % git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")
shenxt@GEN-C02C93JZMD6R github_test3 %
```

这时候我们可以使用 vi 或者其他工具打开冲突的文件, 在这就是 readme.txt:



Git 用 <<<<<<<, =======, >>>>>>> 标记出不同分支的内容，我们修改如下后保存:

我们将这些记号删除, 然后保留自己想要保留的修改就可. 然后重新提交.

```
git add .
git commit-m 'remove conflict'
```

这时候就完成了合并. 可以删除 development 分支了.

这时候使用 git log 也可以看到分支的提交情况. 使用 git log --graph 可以观察分支合并图.

2.6.8 分支策略

在实际开发中，我们应该按照几个基本原则进行分支管理:

首先，master 分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活;

那在哪干活呢? 干活都在 dev 分支上，也就是说，dev 分支是不稳定的，到某个时候，比如 1.0 版本发布时，再把 dev 分支合并到 master 上，在 master 分支发布 1.0 版本;

你和你的小伙伴们每个人都在 dev 分支上干活，每个人都有自己的分支，时不时地往 dev 分支上合并就可以了。

2.7 远程仓库 (GitHub)

上面的所有命令都是针对本地仓库的操作. 当我们希望多个人来协作时, 会将代码发布到一个统一的远程仓库, 然后多个人在本地操作以后, 再推送到远程仓库. 其他人协作时, 需要先同步远程仓库的内容, 再推送自己的修改. 这就要用到了我们最常用的 github.

2.7.1 添加到远程仓库

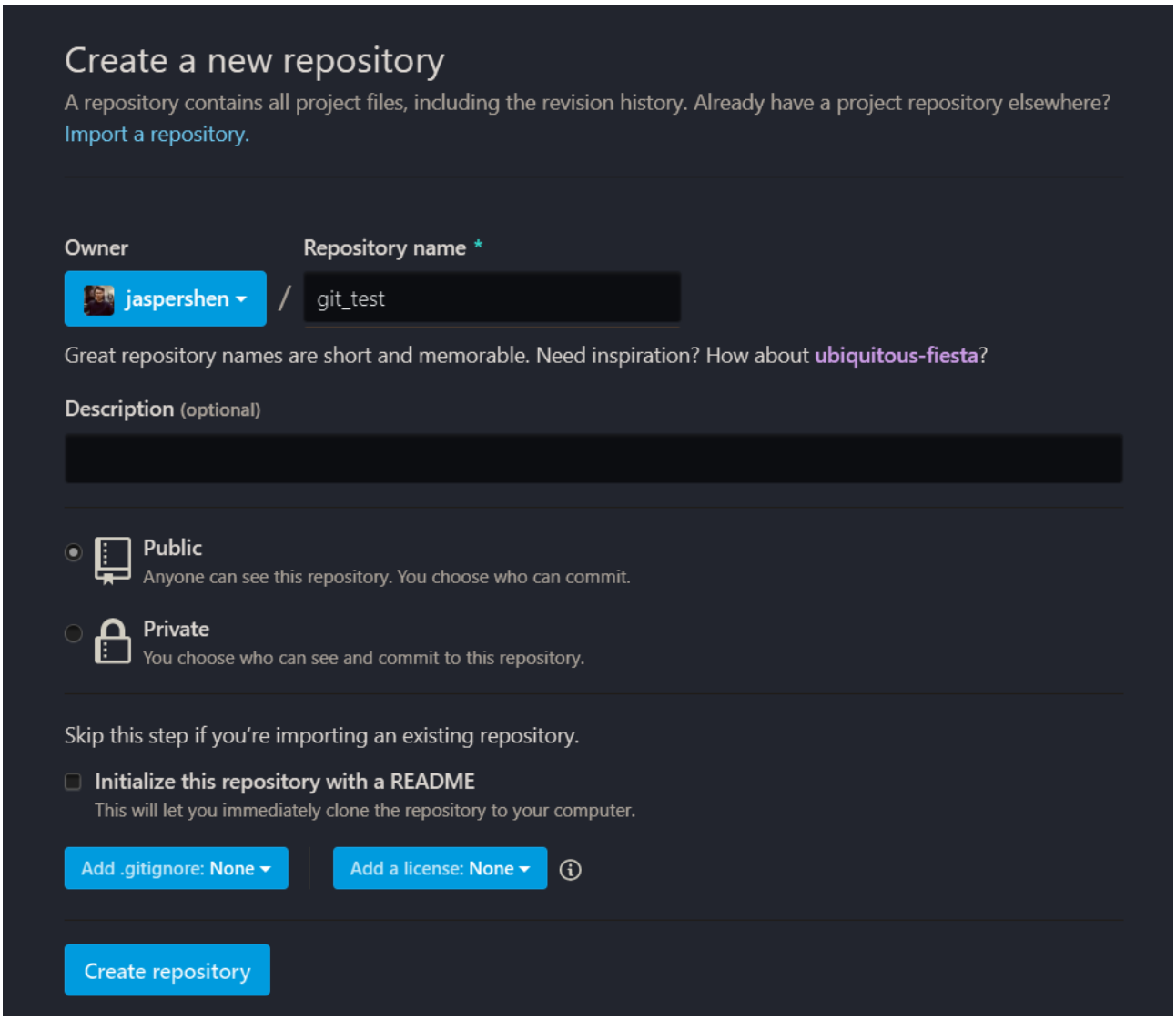
比如我们上面所建立的 `git_test` 本地仓库, 我们需要将其添加到 github 的远程仓库中.

```
git remote add origin your_remote_git_repo # 为本地仓库添加远程仓库
```

其中 `your_remote_git_repo` 是在 github 中创建的. 我们先创建一下, 然后进行尝试.

格式为 `git@github.com:user.name/repo_name`. 当然, 也可以直接使用远程仓库的网址:

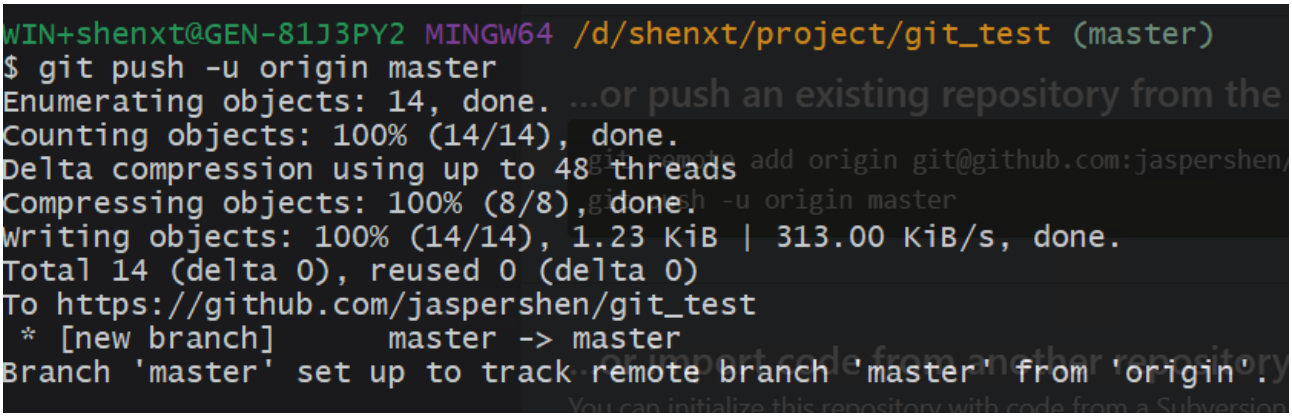
https://github.com/jaspershen/git_test



2.7.2 推送到远程仓库

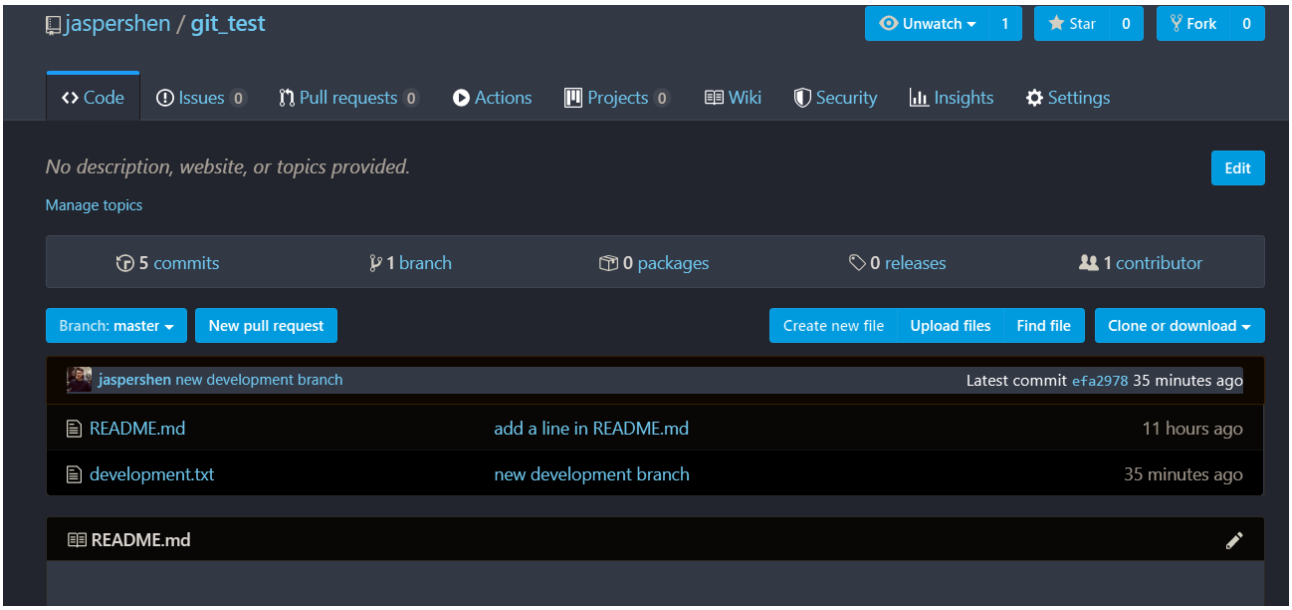
使用 `git push` 将本地仓库的内容推送到远程仓库.

```
git push -u origin master # 第一次推送时使用，可以简化后面的推送或者拉取命令使用
git push origin master # 将本地 master 分支推送到 origin 远程分支，后面之后就可以直接使用该命令推送
```



以后再推送的时候, 就使用 `git push` 就可以了.

这时候可以看到 github 中已经有了内容.



2.7.3 从远程仓库获取最新内容

在多人协作过程中, 当自己完成了本地仓库中的提交, 想要向远程仓库推送前, 需要先获取到远程仓库的最新内容。

可以通过 `git fetch` 和 `git pull` 来获取远程仓库的内容.

```
git fetch origin master
git pull origin master
```

两者之间的区别:

- `git fetch` 是仅仅获取远程仓库的更新内容, 并不会自动做合并.
- `git pull` 在获取远程仓库的内容后, 会自动做合并, 可以看成 `git fetch` 之后 `git merge`. 所以更推荐这个命令.

2.7.4 查看远程仓库信息

```
git remote -v
```

2.7.5 从远程仓库克隆

如果你本地没有仓库, 希望从已有的远程仓库上复制一份代码, 那么你需要 `git clone`.

```
git clone https://github.com/jaspershen/git_test # 通过 https 协议, 克隆 Github 上 git 仓库的源码
git clone jaspershen@github.com/git/git_test # 通过 ssh 协议, 克隆 Github 上 git 仓库的源码
```

2.8 设置忽略文件

有时候仓库中的某些文件或者文件夹并不需要进行同步, 这时候可以通过设置忽略文件`.gitignore` 进行管理.

参考帖子:

https://github.com/onlynight/ReadmeDemo/tree/master/Readmes/GitIgnore

2.9 多人协作

2.9.1 创建远程仓库

在 github 创建远程仓库或者将本地仓库推送到远程仓库. 这里就不在演示了.

2.9.2 本地创建分支并推送到远程仓库

比如我们在 `shenxt` 账户上有一个远程仓库, 创建一个 `jaspershen` 分支.

```
git branch jaspershen
```

然后将该分支推送到远程仓库.

```
git push origin jaspershen
```

这时候再去远程仓库就可以看到 `jaspershen` 分支了.

2.9.3 邀请合作者

在 github 远程仓库的 `setting` 中, 点击 `Manage access`, 然后点击 `invite collarbortor`, 填入邀请人的 ID 即可. 这时候邀请人可以在自己 github 中看到邀请, 同意即可.

2.9.4 合作者参与项目

合作者同意邀请之后, 需要克隆项目, 创建本地分支.

2.9.5 克隆项目

```
git clone git@github_jaspershen:shenxt/github_test4.git
cd github_test4
```

这时候查看分支,

```
git branch
```

会看到只有 master 分支. 所以要创建远程仓库的分支到本地.

```
git checkout -b jaspershen origin/jaspershen
```

这时候再查看分支, 就会在 `jaspershen` 分支了. 这时候合作者 jaspershen 就可以在该分支下进行工作了.

2.9.6 开始工作

这时候合作者 `jaspershen` 可以在分支下正常工作, 然后提交, 合并, 并推送到远程仓库.

- 创建文件

```
touch heloworld.txt
```

然后在该文件下写入一行文字:

```
test from jaspershen
```

- 提交

然后加入到暂存区并提交.

```
git add helloworld.txt
git commit -m "test from jaspershen"
```

- 合并分支

```
git checkout master
git merge jaspershen
```

- 推送到远程仓库

然后可以将其推送到远程分支 (包括 master 和 jaspershen 分支). 注意是直接推送到 `shenxt` 账户的远程仓库.

```
git push origin master
git push origin jaspershen
```

这时候就可以在 shenxt 的远程仓库下看到 jaspershen 的提交工作了.

2.9.7 有冲突怎么办

多人协作是不免会有冲突, 冲突的主要原因就是对同一文件的修改, 所以最好是提前说好不要对同一个文件进行修改, 如果有冲突, 请参考:

https://www.liaoxuefeng.com/wiki/896043488029600/900004111093344

2.9.8 抓取分支

多人协作时，大家都会往 master 分支上推送各自的修改。如果两个人同时对同一个文件进行了修改, 并且同试图推送到 master 分支, 那么就会冲突, 报错. 这时候, 应该先使用 git pull 命令将最新的提交从 origin/master 下抓取下来, 然后本地合并, 解决冲突, 再推送.

比如我们在 shenxt 账户下也创建一个 helloworld.txt 文件. 然后提交, 推送到远程仓库.

```
git push origin master
```

会得到下面的报错:

```
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github_shenxt:shenxt/github_test4.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

我们先使用 git pull 抓取分支的最新提交.

```
git pull
```

也失败了, 因为没有指定本地分支与远程 origin/xxx 分支的链接，根据提示，设置本地分支和远程分支的链接:

```
git branch --set-upstream-to=origin/master master
git branch --set-upstream-to=origin/jaspershen jaspershen
```

然后在 pull.

这回 git pull 成功，但是合并有冲突，需要手动解决，解决的方法和分支管理中的解决冲突完全一样。解决后，提交，再 push:

2.9.9 多人协作的工作模式

1. 首先，可以试图用 git push origin <branch-name> 推送自己的修改;
2. 如果推送失败，则因为远程分支比你的本地更新，需要先用 git pull 试图合并;
3. 如果合并有冲突，则解决冲突，并在本地提交;
4. 没有冲突或者解决掉冲突后，再用 git push origin <branch-name> 推送就能成功!

5. 如果 `git pull` 提示 `no tracking information`，则说明本地分支和远程分支的链接关系没有创建，用命令 `git branch --set-upstream-to <branch-name> origin/<branch-name>`。

这就是多人协作的工作模式，一旦熟悉了，就非常简单。

2.10 How to work together use git and github for metID (for Han)

2.10.1 Set up you git and github

Make sure you have installed github desktop in you mac.

Open you terminal, and then type this code to set up you local git account:

```
git config --global user.name your_github_user_name
git config --global user.email your_github_user_email
```

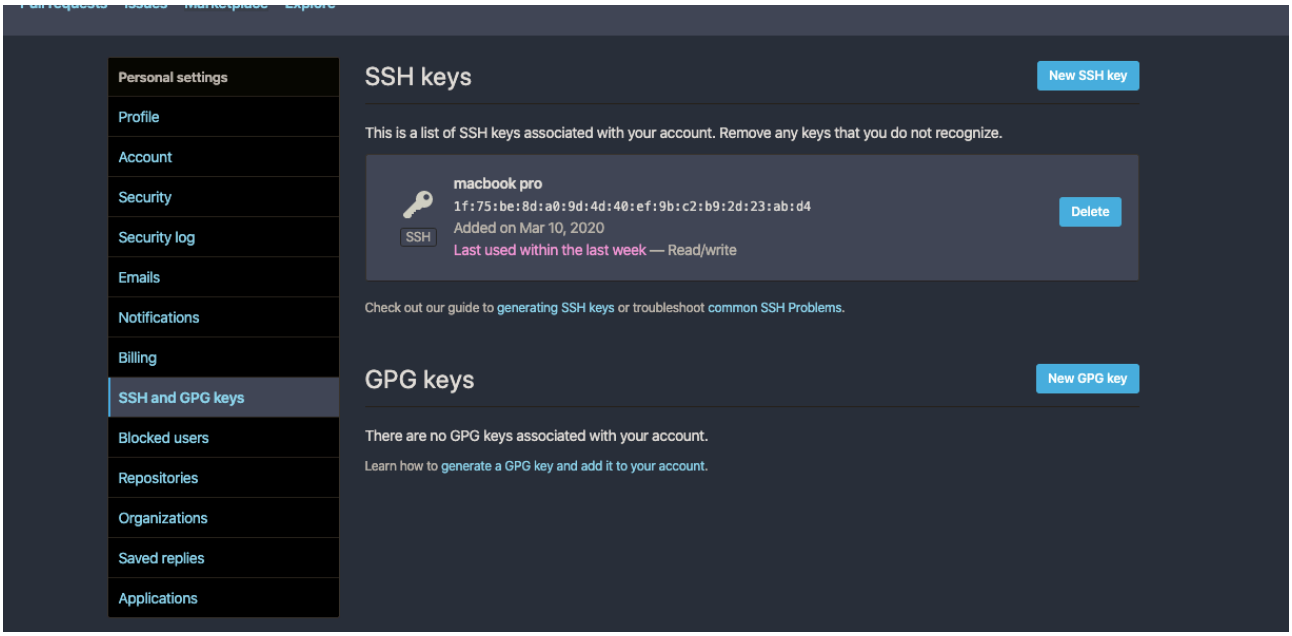
After you set up your github account, then try to confirm if you are successful:

```
git config --global user.name
git config --global user.email
```

Then, you should set up a SSH with you local git and github, still in your terminal:

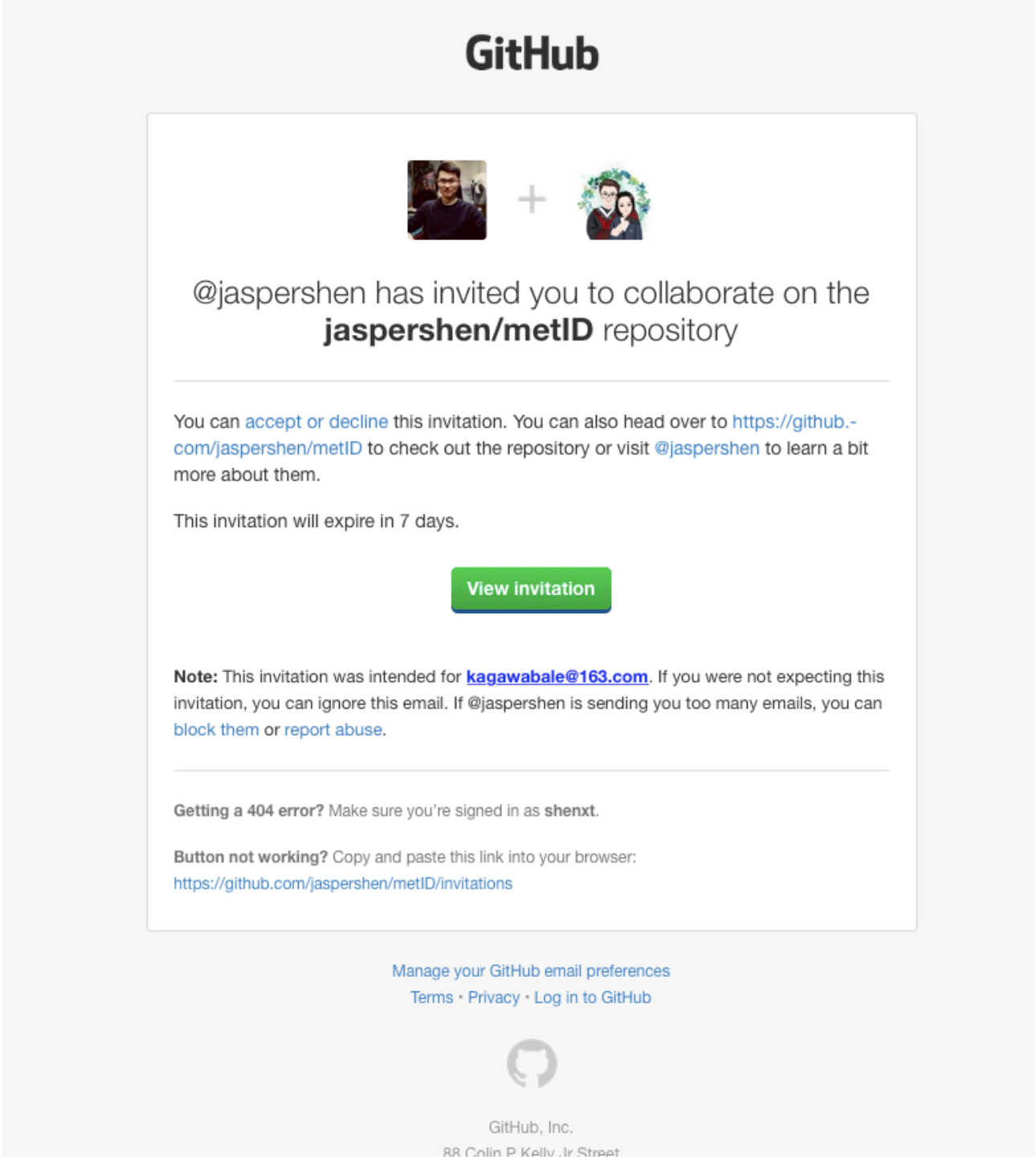
```
#chang directory
cd ~/.ssh
ssh-keygen -t rsa -C "your_github_email"
```

Then open your `/ssh` folder, and you will see there is a `id_rsd.pub`. Open this file and copy all the content. Go to you github website, in the right top concer, cliker your picture, and then select `Settings`. Then select `SSH and GPG keys`, and then click `New SSH key`. For the `Title`, just type a name you want, and then paste the keys you copied from `id_rsd.pub` to the key area. And then click `Add SSH key`.



2.10.2 Accept the invitation

I will send a collaborator invitation email to Han, you will receive this email and please recept the inviation.



2.10.3 Git clone the repository

Open you terminal, and then set to the work directoy you want to keep the metID. Then type this code to clone the metID repo:

```
git clone git@github.com:jaspershen/metID.git
```

Then change you work directory into `metID` folder:

```
cd metID
```

Then check you branch:

```
git branch
```

You will find you only have the master branch. Then chnage to `hanyah` branch:

```
git checkout -b hanyah origin/hanyah
```

Then you can see that you are in branch already:

```
gitbranch
```

2.10.4 Add the repo into the github desktop

Open you github desktop. And open you `Preferences`, and then sign in.

Then click `File`, and select `Add Local Repository`, and then choose the local `metID` package in your mac.

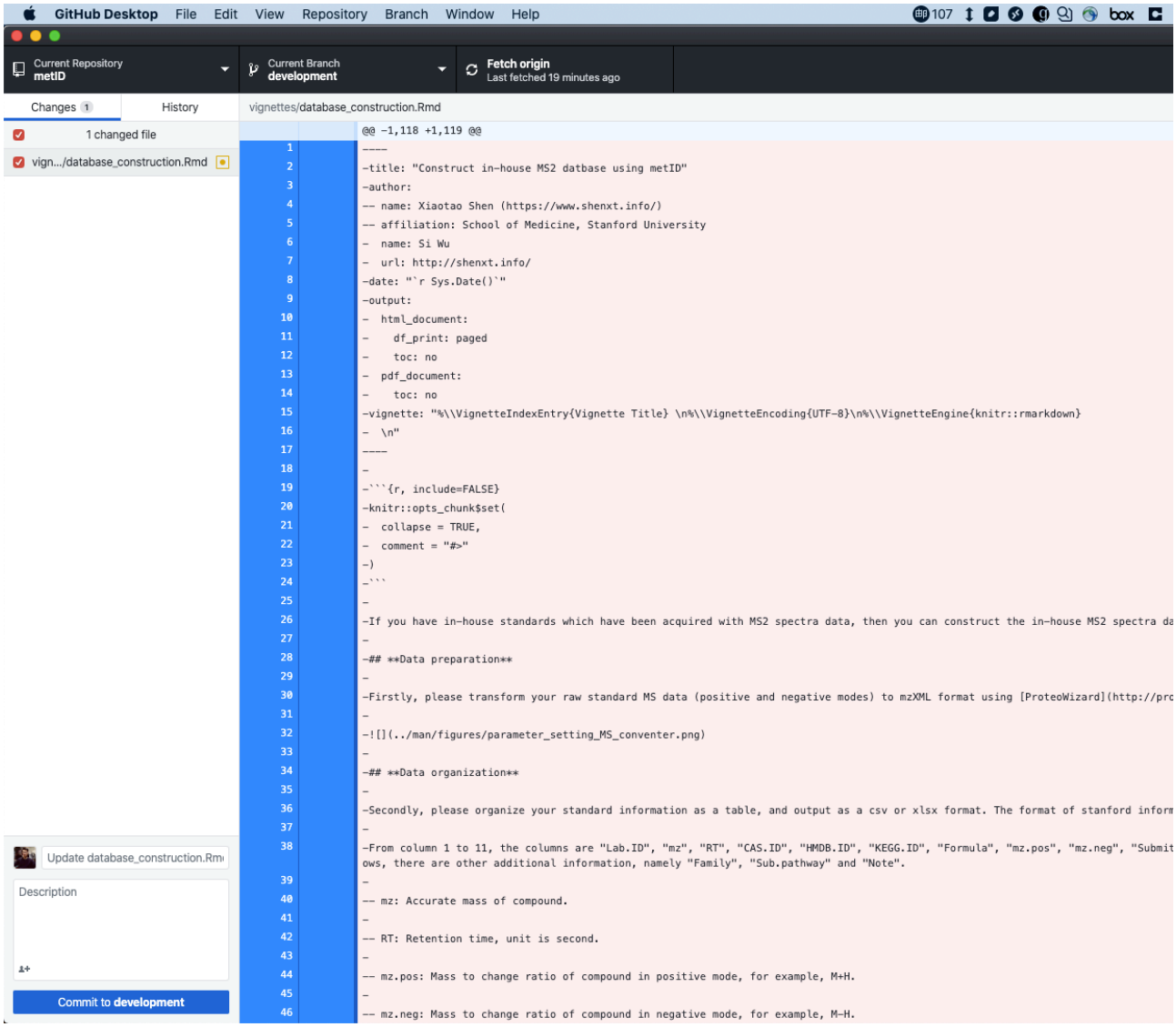
Now you can use github desktop to manage your metID repository.

2.10.5 Begin to work

Now please open your Rstudio, and open the `metID` project. Now you can work with metID.

2.10.6 Push your work to github

After your finish a file or document, and then you can open your github desktop, you can see what you have changed in your documents or files. And then please type some note in your summary, and then click `Commit to hanyah`. And then push them to github by click `Fetch origin`.



Please enable JavaScript to view the comments powered by Disqus.

第 3 章 常用学习链接

- Magit and Git
  1. 如何在 Emacs 中使用 Magit 管理 Git 项目
  2. Magit 使用技巧
  3. 认识 Magit
  4. GitHub 入门指南
- Emacs
  1. 专业 Emacs 入门教程
  2. Emacs 光标移动
  3. Emacs Academic Tools
- MacPorts
  1. Mac Ports Guide

3.1 Quarto Workflows

1. Quarto workflows
2. Quarto：Markdown 又何必只是 Markdown
3. Awesome Quarto Awesome
4. Use R to generate a Quarto blogpost
5. Andrew Heiss's Homepage
6. Adding a blog to your existing Quarto website
7. R 语言空间数据分析实战
8. quarto-website-tutorial ## 其他
9. 毫末科技培训资料

第 4 章 相关问题

1. 在 emacs 中运行 R 出现下面警告信息

```
During startup - Warning messages:
1: Setting LC_CTYPE failed, using "C"
2: Setting LC_COLLATE failed, using "C"
3: Setting LC_TIME failed, using "C"
4: Setting LC_MESSAGES failed, using "C"
5: Setting LC_MONETARY failed, using "C"
```

在.emacs 中添加如下配置可解决

```
(unless (getenv "LC_ALL") (setenv "LC_ALL" "en_US.UTF-8"))
```

2.Mac OS M1 系统在 R 中安装 VGAM 包出现下面问题

```
make: /opt/R/arm64/bin/gfortran: No such file or directory
make: *** [mvt.o] Error 1
```

解决方法如下：

- 利用 Homebrew 安装 gcc
  `brew install gcc`
- 在 Home 文件夹下创建文件 `~/.R/Makevars`

  ```
  mkdir -p ~/.R
  touch ~/.R/Makevars
  ```

- 添加如下信息至文件 `~/.R/Makevars`

  ```
  FC = /opt/homebrew/Cellar/gcc/12.2.0/bin/gfortran
  F77 = /opt/homebrew/Cellar/gcc/12.2.0/bin/gfortran
  FLIBS = -L/opt/homebrew/Cellar/gcc/12.2.0/lib/gcc/12
  ```

第 5 章 路由器

5.1 备忘录

- ROS: admin@82.1+!1#1
- ESXI: root@82.25+Z!2349
- OpenWrt: root@82.3+!0m0
- VPN L2TP: xzhang@!1@1
- Docker Ports
  - NAS: admin@82.245:1819
  - Calibre: admin@NAS:8083+!0m0
  - qBittorrent:admin@NAS:8085+!0@0
  -

5.2 Router OS 动态更新阿里域名云解析

主要是利用 vercel 和 github，采用 PHP 语言实现

5.2.1 创建访问控制 RAM 的 AccessKey

这部分内容主要参考博文RouterOS 利用 aliyun 的 API 接口实现 DDNS 动态解析中的第一部分内容。

1、阿里云网站 –> 产品 –> 安全 –> 应用身份服务，这个就是控制 API ，用户管理，新建用户，填写用户名和勾上” 为该用户自动生成 AccessKey”，保存好这个 accesskey。

2、策略管理 –> 自定义授权策略，新建授权策略，选择空白模版，授权策略名称随便填（如 alidns），策略内容为下面的内容（修改下面内容中的域名为你自己要做 DDNS 的域名）

action 是 api 的接口，只接受 AddDomainRecord（增加域名解析），DescribeDomainRecords（输出域名解析列表）和 UpdateDomainRecord（修改域名解析记录）

Resource 是指被授权的具体对象，这边 domain/abc.com 需要修改成你自己的域名 domain/xxx.com。这样就是授权对象是该域名

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "alidns:AddDomainRecord",
        "alidns:DescribeDomainRecords",
        "alidns:UpdateDomainRecord"
      ],
      "Resource": "acs:alidns:*:*:domain/abc.com",
      "Effect": "Allow"
```

```
    }
  ]
}
```

3、用户管理，对上面创建的用户，点击授权，选择刚才自定义创建的策略，确定。

5.2.2 PHP 制作 aliyun API 动态解析接口

- PHP 源码 Git 仓库创建
  1. 利用xinzhangseu/vercel-php仓库模板在自己的 github 账户新建仓库
  2. 将 api 文件夹中的 index.php 更改为下面内容（源码来自77bx/alidns-api-php），并将”Location: //xxxxx.xxxx.com/” 改为后面自己 vercel 生成的网址即可。

```
<?php
/***
* Alidns-api-php V1.3
* By Star.Yu
***/
if($_SERVER['REQUEST_METHOD']=="POST"){
  $request = $_POST;
}
if($_SERVER['REQUEST_METHOD']=="GET"){
  $request = $_GET;
}
if(is_array($request)&&count($request)<1){
  Header("Location: //xxxxx.xxxx.com/");
  exit('2');
}

if(empty($request['id'])){
  exit('2');
}elseif(empty($request['secret'])){
  exit('2');
}elseif(empty($request['domain'])){
  exit('2');
}elseif(empty($request['record'])){
  exit('2');
}else{
  $ip = empty($request['ip']) ? $_SERVER['REMOTE_ADDR'] : addslashes($request['ip']);
  $accessKeyId = addslashes($request['id']);
  $accessKeySecret = addslashes($request['secret']);
  $record = addslashes($request['record']);
  $domain = addslashes($request['domain']);
  $type = empty($request['type']) || $request['type']!='AAAA' ? 'A' : addslashes($request['type']);
}
```

```
if($type === 'A' && !filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV4)){
  exit('1');
}
if($type === 'AAAA' && !filter_var($ip, FILTER_VALIDATE_IP, FILTER_FLAG_IPV6)){
  exit('1');
}

//公共参数 Timestamp GMT 时间
$Timestamp = gmdate('Y-m-d\TH:i:s\Z',time());

//Signature percentEncode 函数
function percentEncode($str) {
  $res = urlencode($str);
  $res = str_replace(array('+', '*'), array('%20', '%2A'), $res);
  $res = preg_replace('/%7E/', '~', $res);
  return $res;
}

//唯一数，用于防止网络重放攻击
function generateByMicrotime() {
  $microtime = microtime(true);
  $microtime = str_replace('.', '', $microtime);
  return $microtime;
}

//sign
function sign($parameters, $accessKeySecret){
  ksort($parameters);
  $canonicalizedQueryString = '';
  foreach ($parameters as $key => $value) {
    $canonicalizedQueryString .= '&' . percentEncode($key) . '=' . percentEncode($value);
  }
  $stringToBeSigned = 'POST&%2F&' . percentEncode(substr($canonicalizedQueryString, 1));
  $signature = base64_encode(hash_hmac('sha1', $stringToBeSigned, $accessKeySecret. '&', true));
  return $signature;
}

function geturl($public, $request, $accessKeySecret){
  $params = array_merge($public, $request);
  $params['Signature'] =  sign($params, $accessKeySecret);
  $uri = http_build_query($params);
  $url = 'https://alidns.aliyuncs.com/?'.$uri;
  return $url;
```

```
}

function ssl_post($url){
  $curl = curl_init();
  curl_setopt($curl, CURLOPT_URL, $url);
  curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, 0);
  curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, 0);
  curl_setopt($curl, CURLOPT_POST, 1);
  curl_setopt($curl, CURLOPT_TIMEOUT, 30);
    curl_setopt($curl, CURLOPT_HEADER, 0);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
  $tmpInfo = curl_exec($curl);
  if (curl_errno($curl)) {
     echo 'Errno'.curl_error($curl);
  }
  curl_close($curl);
  return $tmpInfo;
}

$public = array(
  'Format'    =>  'json',
  'Version' =>    '2015-01-09',
  'AccessKeyId'   =>  $accessKeyId,
  'SignatureMethod'   =>  'HMAC-SHA1',
  'Timestamp' =>  $Timestamp,
  'SignatureVersion'  =>  '1.0',
  'SignatureNonce'    =>  generateByMicrotime()
  );
$search = array(
  'Action'    =>  'DescribeDomainRecords',
  'DomainName'    =>  $domain,
  'PageSize' => '500',
  'RRKeyWord' => $record,
  'Type' => $type
  );

//搜索 record 相关的记录列表
$data = json_decode(ssl_post(geturl($public,$search, $accessKeySecret)),true);

if(empty($data['DomainRecords'])){
  exit('1');
}else{
  foreach($data['DomainRecords']['Record'] as $value){
    $record_arr = array();
```

```
    if($value['RR'] == $record){
      $record_id = $value['RecordId'];
      $record_arr = $value;
      break;
    }
  }

  if(empty($record_id)){
    $add = array(
      'Action'    =>  'AddDomainRecord',
      'DomainName'    =>  $domain,
      'RR'    =>  $record,
      'Type'    =>  $type,
      'Value'    =>  $ip,
      'TTL'    =>  '600',
    );
    $data = json_decode(ssl_post(geturl($public,$add, $accessKeySecret)),true);
    if(empty($data['RecordId'])){
      exit('1');
    }else{
      exit('0');
    }
  }else{
    if($record_arr['Value'] == $ip){
      exit('0');
    }else{
      $edit = array(
        'Action'    =>  'UpdateDomainRecord',
        'RecordId'    =>  $record_id,
        'RR'    =>  $record,
        'Type'    =>  $type,
        'Value'    =>  $ip,
        'TTL'    =>  '600',
      );
      $data = json_decode(ssl_post(geturl($public,$edit, $accessKeySecret)),true);
      if(empty($data['RecordId'])){
        exit('1');
      }else{
        exit('0');
      }
    }
  }
}
```

3.

5.2.3 ROS script

Ros 新建 Script，代码如下：

```
#aliyun Access Key
:local id " 阿里云的 AccessKey ID"
:local secret " 阿里云的 AccessKey secret"

#domain
:local domain "abc.com"# 更改为自己的需要 ddns 的域名
:local record "@"     # @ 或者 www

#PPPoE-out
:local pppoe "pppoe-out1"

:global aliip "8.8.8.8" # 随机填写的 ip 地址，运行 script 后会更新为家庭网络的公网 ip
:local ipaddr [/ip address get [/ip address find interface=$pppoe] address]
:set ipaddr [:pick $ipaddr 0 ([len $ipaddr] -3)]

:if ($ipaddr != $aliip) do={
  :log info "[Cloudflare DDNS] WAN IPv4 address for interface $wanif has been changed to $ip4new."
  :local result [/tool fetch url="https://aliyun-ddns-api.vercel.app/api/index.php?id=$id&secret=$se
:if ($result->"status" = "finished") do={
:if ($result->"data" = "0") do={
:set aliip $ipaddr
:log info "alidns update ok";
} else={
:log info "alidns update error";
}
}
}
```

5.3 相关链接

1. RouterOS 配合 PHP 实现动态更新阿里域名云解析
2. RouterOS 利用 aliyun 的 API 接口实现 DDNS 动态解析
3. Examples & Sample Projects for Vercel
4. PHP Runtime for Vercel

第 6 章 Summary

In summary, this book has no content whatsoever.

```
1 + 1
```

```
[1] 2
```

References