工作流程，常用命令，托管服务，使用

工作流程，常用命令，托管服务，使用idea操作gir

、、、git add 工作区进入暂存区

、、、git status 查看文件状态

、、、git add . 所有文件进入暂存区

、、、git commit -m "name1" 暂存区进入缓存区,版本命名

、、、git log 看日志

在大型开源项目里，查看多人对项目的修改情况

git log --pretty=oneline --abbrev-commit --all --graph --decorate

- 命令形式：git log [option]
  - options
    - --all 显示所有分支
    - --pretty=oneline 将提交信息显示为一行
    - --abbrev-commit 使得输出的commitId更简短
    - --graph 以图的形式显示

回退版本

git reset --hard commitID

直接git log 查看ID，点击就可以复制，然后按滚轮就可以粘贴

```
chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git-log
* bf1a196 (HEAD -> master) 2
* 362693d 1

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git reset 362693d
Unstaged changes after reset:
M       笔记.txt

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ ^C

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git reset -hard 362693d
error: did you mean `--hard` (with two dashes)?

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git reset --hard 362693d
HEAD is now at 362693d 1

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$
```

git reflog 记录所有修改日志

```
chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git reflog
362693d (HEAD -> master) HEAD@{0}: reset: moving to 362693d
362693d (HEAD -> master) HEAD@{1}: reset: moving to 362693d
bf1a196 HEAD@{2}: commit: 2
362693d (HEAD -> master) HEAD@{3}: commit (initial): 1
```

git 忽略特定文件的管理

```
chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ touch .gitignore

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ ll
total 29
drwxr-xr-x 1 chw 197121   0  5月 21 21:56 ./
drwxr-xr-x 1 chw 197121   0  5月 21 21:29 ../
drwxr-xr-x 1 chw 197121   0  5月 21 21:54 .git/
-rw-r--r-- 1 chw 197121   0  5月 21 21:56 .gitignore
-rw-r--r-- 1 chw 197121 183  5月 21 21:54 笔记.txt
```

git branch 查看分支

git branch dev01 创建新分支

工作区只能对一个分支起作用

、、、、git checkout dev01 切换分支

、、、、git checkout -b dev02 创建并切换分支

```
chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git-log
* 362693d (HEAD -> master, dev1) 1

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git branch
  dev1
* master

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git checkout  -b dev02
Switched to a new branch 'dev02'

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (dev02)
$
```

```
chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (dev02)
$ git add .

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (dev02)
$ git commit -m "2"
[dev02 a74c053] 2
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore
 create mode 100644 addtest.txt

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (dev02)
$ git-log
* a74c053 (HEAD -> dev02) 2
* 362693d (master, dev1) 1

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (dev02)
$ git checkout master
Switched to branch 'master'

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ touch a.docx

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git add .

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git commit "213"
error: pathspec '213' did not match any file(s) known to git

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git commit -m "213"
[master 8ba3845] 213
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.docx

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git-log
* 8ba3845 (HEAD -> master) 213
| * a74c053 (dev02) 2
|/
* 362693d (dev1) 1
```

、、、git merge dev02 合并分支

```
chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git merge dev02
Merge made by the 'ort' strategy.
 .gitignore  | 0
 addtest.txt | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 .gitignore
 create mode 100644 addtest.txt

chw@DESKTOP-05ILK1O MINGW64 ~/Desktop/test (master)
$ git-log
*   41bb809 (HEAD -> master) Merge branch 'dev02'
|\
| * a74c053 (dev02) 2
* | 8ba3845 213
|/
* 362693d (dev1) 1
```

3.4.7、删除分支

不能删除当前分支，只能删除其他分支

git branch -d b1 删除分支时，需要做各种检查

git branch -D b1 不做任何检查，强制删除

解决冲突：同一文件的同一行，不同的修改

```
zhang@zhangmeng MINGW64 ~/Desktop/git-test01 (master)
$ git merge dev
Auto-merging file01.txt
CONFLICT (content): Merge conflict in file01.txt
Automatic merge failed; fix conflicts and then commit the result.
```

```
<<<<<<< HEAD
update count=3
=======
update count=2
>>>>>>> dev
```

直接改

分支命名规则

- master （生产） 分支

  线上分支，主分支，中小规模项目作为线上运行的应用对应的分支；

- develop （开发） 分支

  是从master创建的分支，一般作为开发部门的主要开发分支，如果没有其他并行开发不同期上线要求，都可以在此版本进行开发，阶段开发完成后，需要是合并到master分支,准备上线。

- feature/xxxx分支

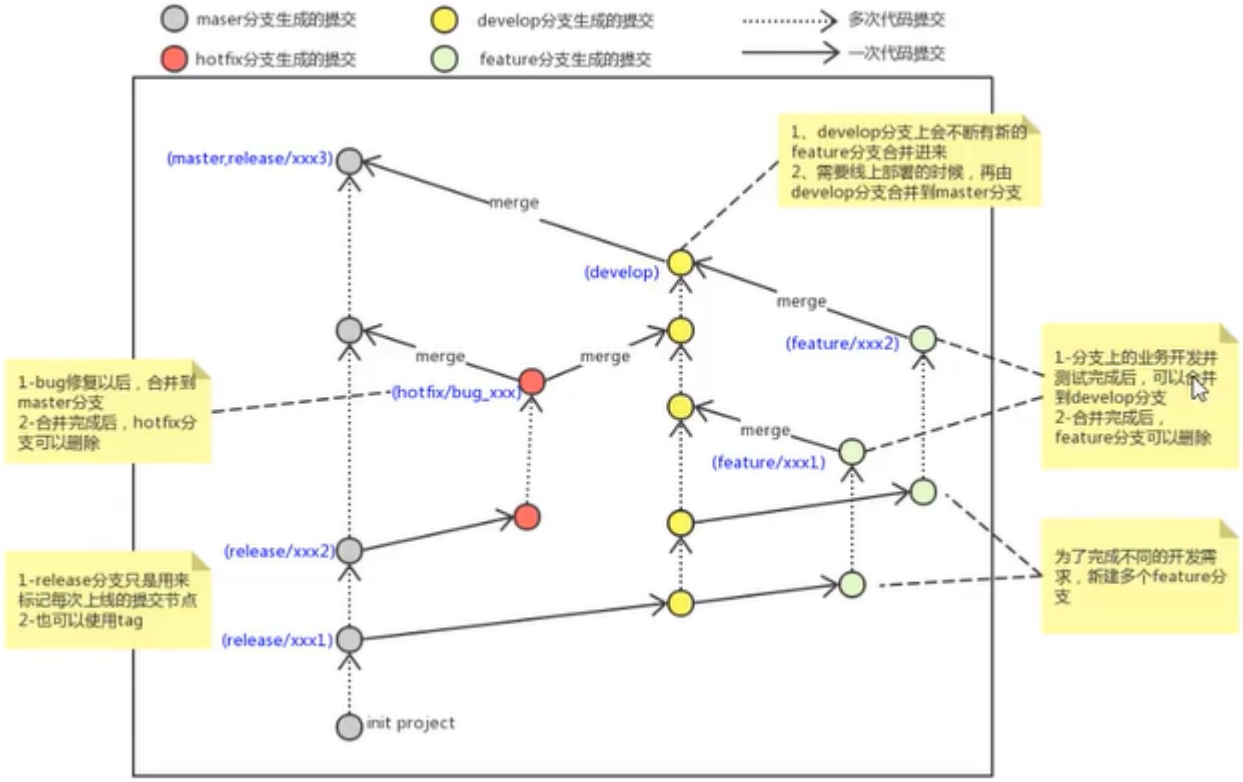
  从develop创建的分支，一般是同期并行开发，但不同期上线时创建的分支，分支上的研发任务完成后合并到develop分支。

- hotfix/xxxx分支，

  从master派生的分支，一般作为线上bug修复使用，修复完成后需要合并到master、test、develop分支。

- 还有一些其他分支，在此不再详述，例如test分支（用于代码测试）、pre分支（预上线分支）等等。

公司使用的规则


maser分支生成的提交
develop分支生成的提交
多次代码提交
hotfix分支生成的提交
feature分支生成的提交
一次代码提交
(master,release/xxx3)
merge
1、develop分支上会不断有新的feature分支合并进来
2、需要线上部署的时候，再由develop分支合并到master分支
(develop)
merge
merge
merge
(feature/xxx2)
1-分支上的业务开发并测试完成后，可以合并到develop分支
2-合并完成后，feature分支可以删除
1-bug修复以后，合并到master分支
2-合并完成后，hotfix分支可以删除
(hotfix/bug_xxx)
merge
(feature/xxx1)
(release/xxx2)
1-release分支只是用来标记每次上线的提交节点
2-也可以使用tag
为了完成不同的开发需求，新建多个feature分支
(release/xxx1)
init project