CodeToolchains Documentation

Release 1.0.0

HappyAnony

Jun 22, 2018

Contents

本手册的定位是实用指南，不求大而全，但求小而精！

CHAPTER 1

目录

1.1 文本编辑器

1.1.1 目录

Sublime

Vim

目录

实用操作

目录

打开文件

文本操作

关闭文件

配置文件

插件管理

目录

Vundle

参考文档:

- 如何在Linux上使用Vundle管理Vim插件

vim-plug

定制vim

目录

定制C/C++开发环境

定制Python开发环境

VSCode

1.2 版本控制器

1.2.1 目录

Git版本控制

目录

git简介

- 功能本质
- 发展简史
- 优势缺陷

0x00 功能本质

0x01 发展简史

0x02 优势缺陷

git原理

git实操

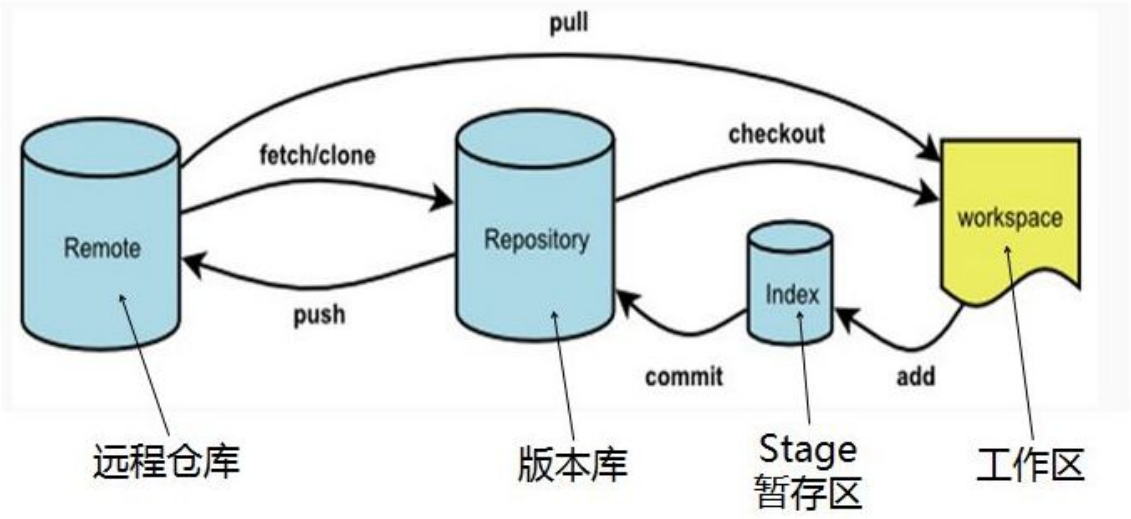
目录

工具安装

获取帮助

基本配置

- git运行前配置
- 自定义git配置



获取仓库

本地仓库维护

远程仓库维护

- github托管
- 自建git服务器
- 分布式git

git高级

- git嵌入到开发环境
- git嵌入到应用
- 其它VCS迁移到git

svn版本控制

1.3 IDE集成开发环境

1.3.1 目录

Eclipse

目录

C/C++开发环境

目录

JDK环境安装

Eclipse是基于Java的可扩展开源开发平台，所以安装Eclipse 前需要确保电脑已安装JDK

JDK的安装可参考：java开发环境配置

Windows下安装java开发环境的一般步骤是：

- 下载*JDK*
- 安装*JDK*
- 配置环境变量
- 验证安装成功

0x00 下载JDK

官方下载地址：JDK下载

在下载页面需要选择接受许可，并根据自己的系统选择对应的版本

0x01 安装JDK

windows上的程序安装最简单，直接一直下一步默认安装就行

安装JDK的时候默认会安装JRE，其实JDK里面已经自带JRE环境，可以不用安装它

安装JDK，安装过程中可以自定义安装目录等信息

0x02 配置环境变量

JDK安装后需要配置系统环境变量，便于随处随时都可以使用JDK环境

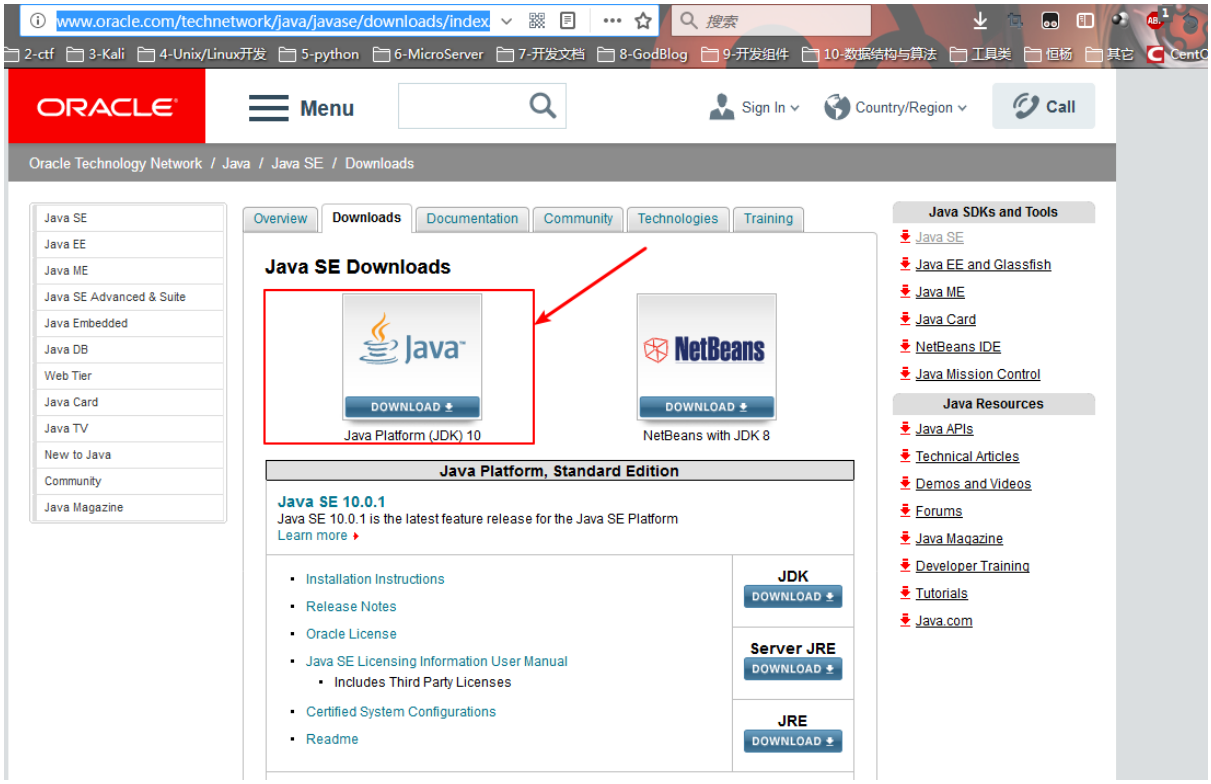
右击我的电脑，点击属性，选择高级系统设置

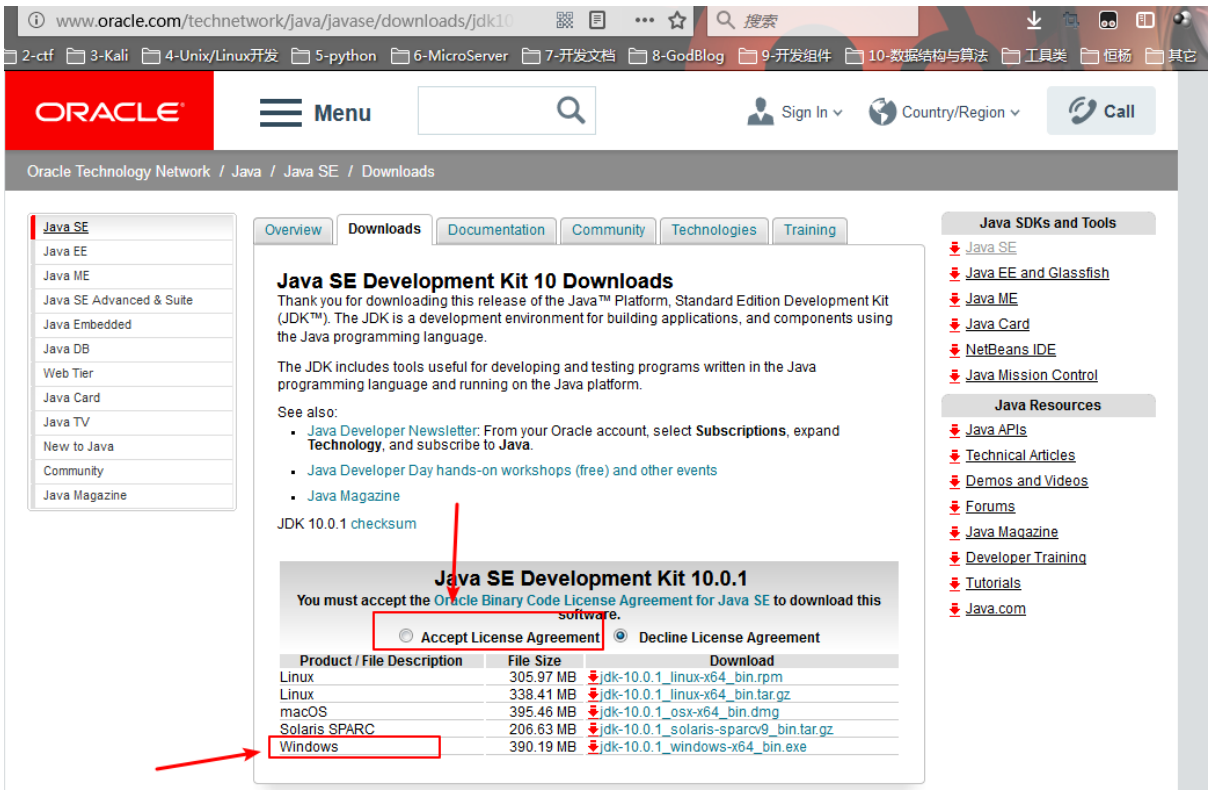
选择高级选项卡，点击环境变量

然后就会出现如下图所示的画面

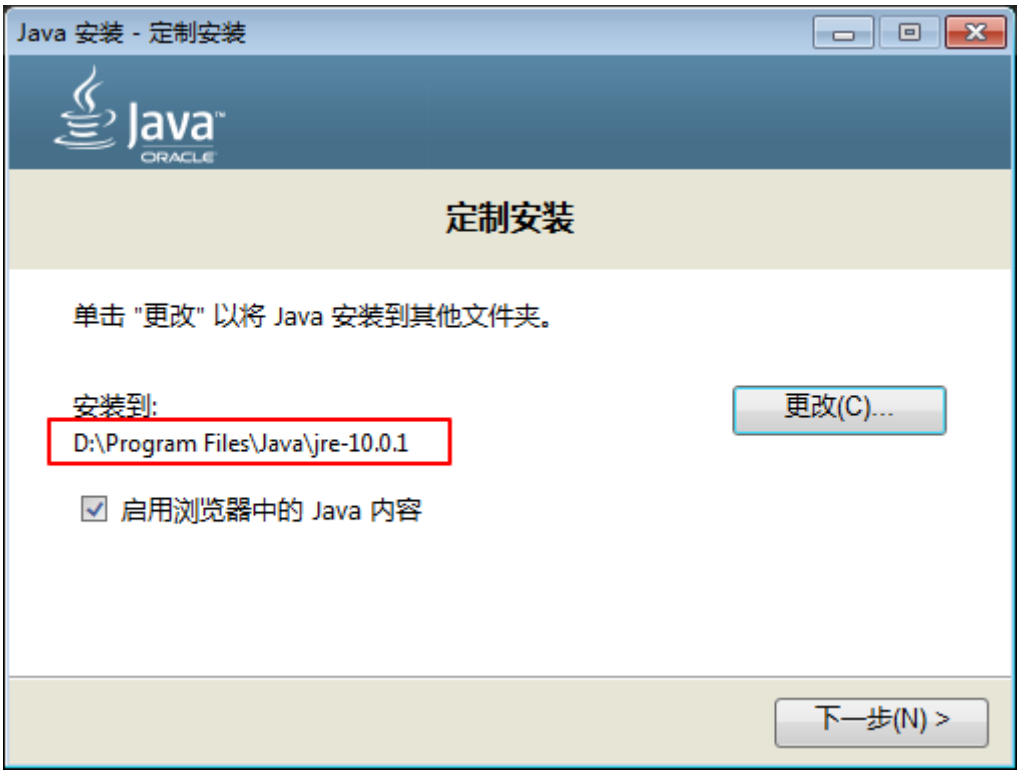
在系统变量中设置2项属性，JAVA_HOME、PATH(大小写无所谓)即可；若已存在则点击编辑，不存在则点击新建

变量设置参数如下

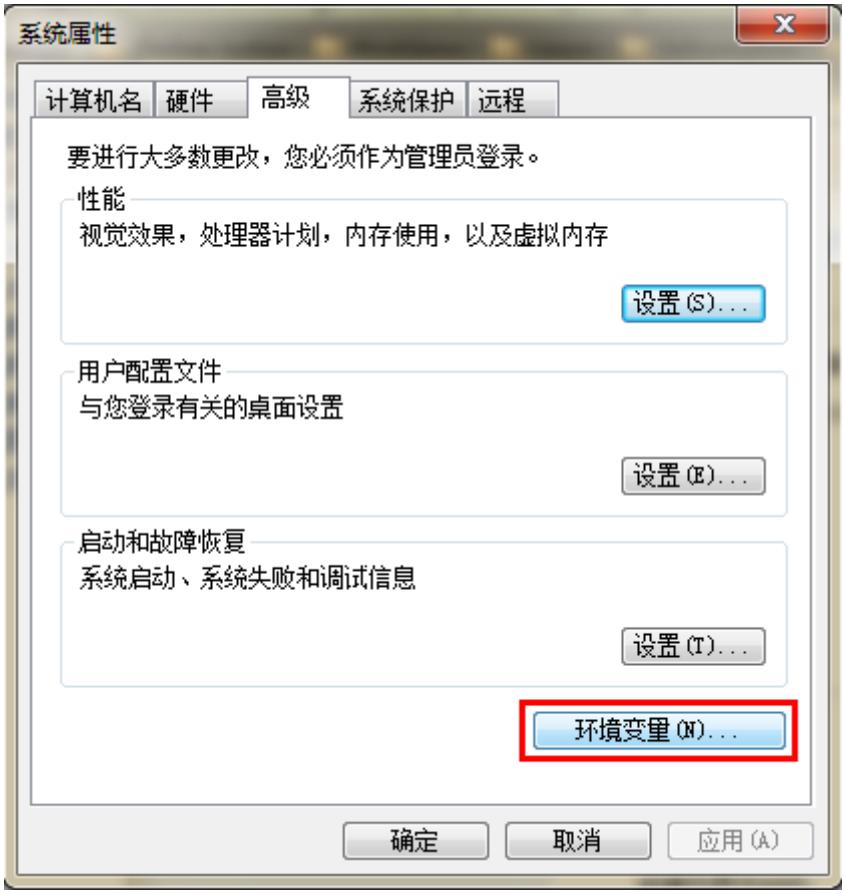

www.oracle.com/technetwork/java/javase/downloads/index
ORACLE
Menu
Sign In
Country/Region
Call
Oracle Technology Network / Java / Java SE / Downloads
Java SE
Java EE
Java ME
Java SE Advanced & Suite
Java Embedded
Java DB
Web Tier
Java Card
Java TV
New to Java
Community
Java Magazine
Overview
Downloads
Documentation
Community
Technologies
Training
Java SE Downloads
Java Platform (JDK) 10
NetBeans with JDK 8
DOWNLOAD
Java Platform, Standard Edition
Java SE 10.0.1
Java SE 10.0.1 is the latest feature release for the Java SE Platform
Learn more
Installation Instructions
Release Notes
Oracle License
Java SE Licensing Information User Manual
Includes Third Party Licenses
Certified System Configurations
Readme
JDK
Server JRE
JRE
Java SDKs and Tools
Java SE
Java EE and Glassfish
Java ME
Java Card
NetBeans IDE
Java Mission Control
Java Resources
Java APIs
Technical Articles
Demos and Videos
Forums
Java Magazine
Developer Training
Tutorials
Java.com

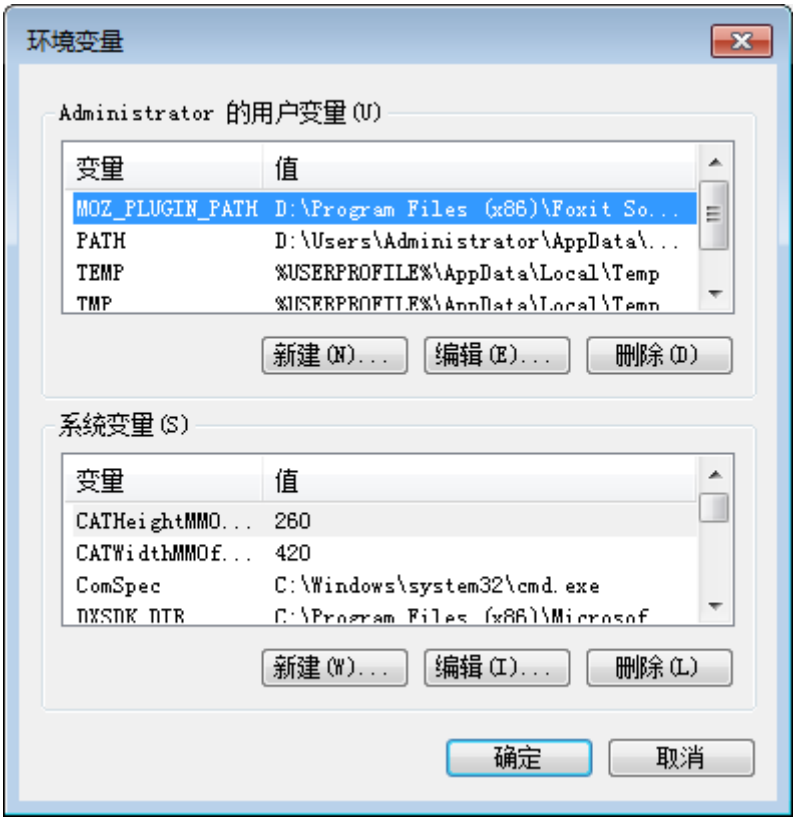

www.oracle.com/technetwork/java/javase/downloads/jdk10
ORACLE
Menu
Sign In
Country/Region
Call
Oracle Technology Network / Java / Java SE / Downloads
Java SE
Java EE
Java ME
Java SE Advanced & Suite
Java Embedded
Java DB
Web Tier
Java Card
Java TV
New to Java
Community
Java Magazine
Overview
Downloads
Documentation
Community
Technologies
Training
Java SE Development Kit 10 Downloads
Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, and components using the Java programming language.
The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.
See also:
Java Developer Newsletter: From your Oracle account, select Subscriptions, expand Technology, and subscribe to Java.
Java Developer Day hands-on workshops (free) and other events
Java Magazine
JDK 10.0.1 checksum
Java SE Development Kit 10.0.1
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.
Accept License Agreement
Decline License Agreement
Product / File Description
File Size
Download
Linux 305.97 MB jdk-10.0.1_linux-x64_bin.rpm
Linux 338.41 MB jdk-10.0.1_linux-x64_bin.tar.gz
macOS 395.46 MB jdk-10.0.1_osx-x64_bin.dmg
Solaris SPARC 206.63 MB jdk-10.0.1_solaris-sparcv9_bin.tar.gz
Windows 390.19 MB jdk-10.0.1_windows-x64_bin.exe
Java SDKs and Tools
Java SE
Java EE and Glassfish
Java ME
Java Card
NetBeans IDE
Java Mission Control
Java Resources
Java APIs
Technical Articles
Demos and Videos
Forums
Java Magazine
Developer Training
Tutorials
Java.com


Java(TM) SE Development Kit 10.0.1 (64-bit) - 定制安装
Java
ORACLE
从下面的列表中选择要安装的可选功能。您可以在安装后使用控制面板中的"添加/删除程序"实用程序更改所选择的功能
功能说明
开发工具
源代码
公共 JRE
独立 JRE。任何应用程序均可使用该 JRE。它将 Java 插件和 Java Web Start 注册到浏览器和系统。可独立于 JDK 进行卸载。它要求硬盘驱动器上有 200MB 空间。
默认安装
自定义路径
安装到:
D:\Program Files\Java\jdk-10.0.1\
更改(C)...
< 上一步(B)
下一步(N) >
取消


Java 安装 - 定制安装
Java
ORACLE
定制安装
单击 "更改" 以将 Java 安装到其他文件夹。
安装到:
D:\Program Files\Java\jre-10.0.1
更改(C)...
启用浏览器中的 Java 内容
下一步(N) >


控制面板 ▸ 系统和安全 ▸ 系统
搜索控制面板
控制面板主页
设备管理器
远程设置
系统保护
高级系统设置
查看有关计算机的基本信息
Windows 版本
Windows 7 旗舰版
版权所有 © 2009 Microsoft Corporation
。保留所有权利。


系统属性
计算机名
硬件
高级
系统保护
远程
要进行大多数更改，您必须作为管理员登录。
性能
视觉效果，处理器计划，内存使用，以及虚拟内存
设置(S)...
用户配置文件
与您登录有关的桌面设置
设置(E)...
启动和故障恢复
系统启动、系统失败和调试信息
设置(T)...
环境变量(N)...
确定
取消
应用(A)



变量名	变量值
JAVA_HOME(新建)	D:\Program Files\Java\jdk-10.0.1(要根据自己的实际路径配置)
Path(编辑)	%JAVA_HOME%\bin;D:\Program Files\Java\jre-10.0.1\bin;

注意:

- 在Windows10中，因为系统的限制，path变量只可以使用JDK的绝对路径；%JAVA_HOME%会无法识别，导致配置失败
- JAVA_HOME变量只是设置被Path变量引用，也可以删除，直接使用绝对路径即可
- Path变量后面的D:\Program Files\Java\jre-10.0.1\bin路径可有可无，因为%JAVA_HOME%\bin已经包含了JRE环境
- 如果使用1.5以上版本的JDK，不用设置CLASSPATH环境变量，也可以正常编译和运行Java程序。

新建JAVA_HOME变量

编辑Path变量

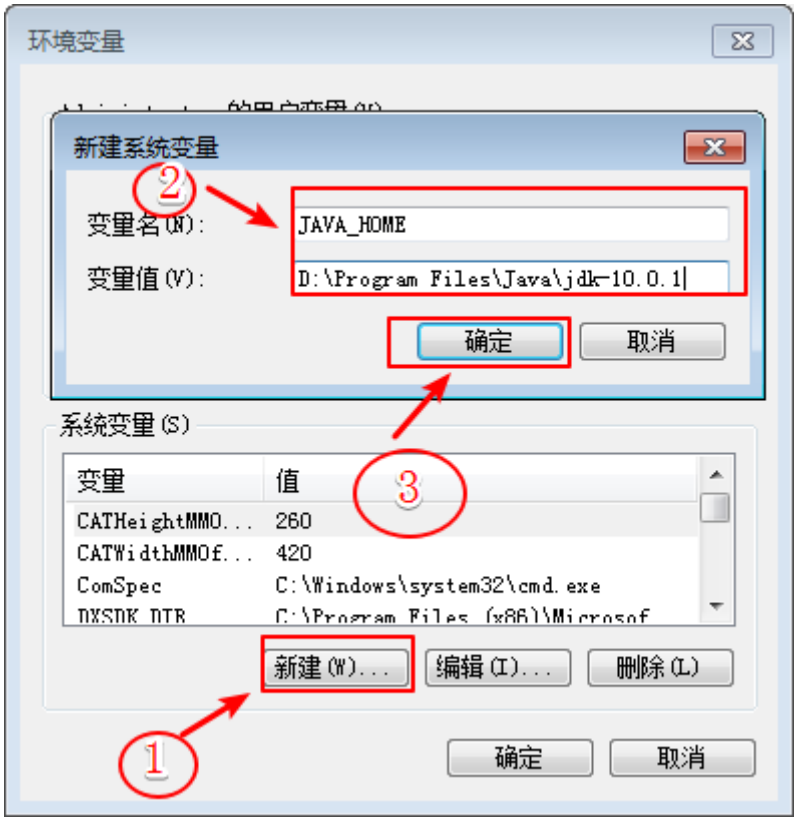
0x03 验证安装成功

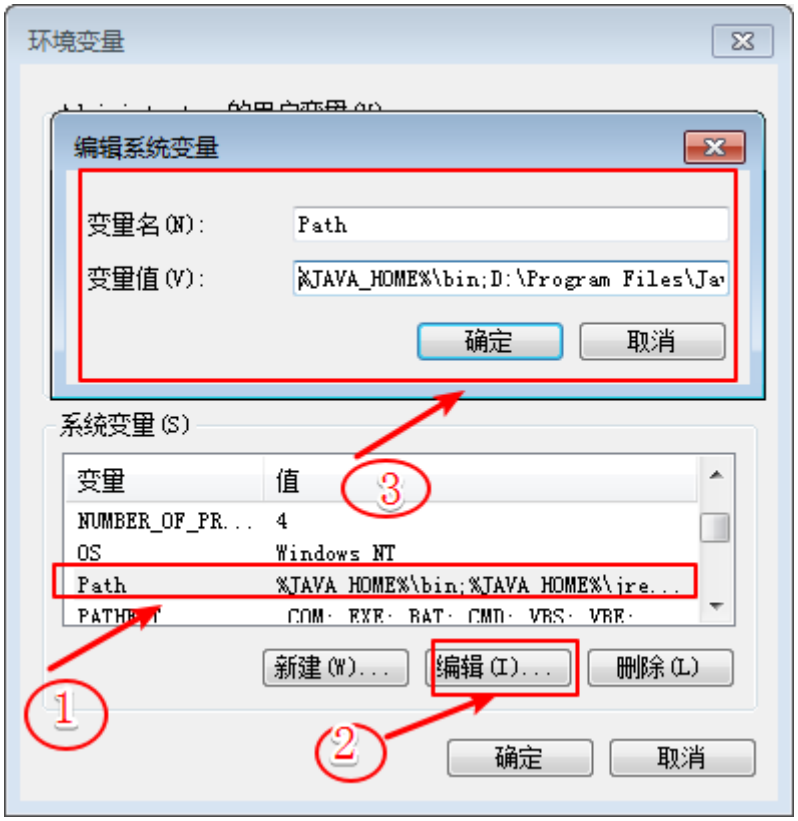
键入命令java --version、java、javac几个命令，能够回显相关信息，说明环境变量配置成功

输入where java命令可以查看java的查找路径有哪些

Eclipse CDT安装

安装完JDK环境后，我们再来安装Eclipse

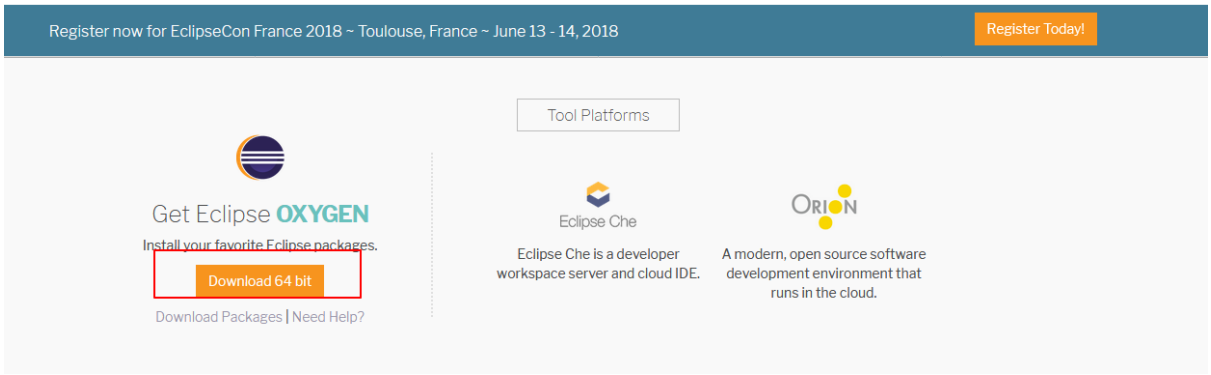

环境变量
新建系统变量
变量名(N):
JAVA_HOME
变量值(V):
D:\Program Files\Java\jdk-10.0.1
确定
取消
系统变量(S)
变量
值
CATHeightMMO...
260
CATWidthMMOf...
420
ComSpec
C:\Windows\system32\cmd.exe
新建(W)...
编辑(I)...
删除(L)
确定
取消
1
2
3


环境变量
编辑系统变量
变量名(N):
Path
变量值(V):
%JAVA_HOME%\bin;D:\Program Files\Ja
确定
取消
系统变量(S)
变量
值
NUMBER_OF_PR...
4
OS
Windows_NT
Path
%JAVA_HOME%\bin;%JAVA_HOME%\jre...
新建(W)...
编辑(I)...
删除(L)
确定
取消
1
2
3

```
C:\Users\Administrator
λ java --version
java 10.0.1 2018-04-17
Java(TM) SE Runtime Environment 18.3 (build 10.0.1+10)
Java HotSpot(TM) 64-Bit Server VM 18.3 (build 10.0.1+10, mixed mode)

C:\Users\Administrator
λ java
用法: java [options] <主类> [args...]
           (执行类)
   或  java [options] -jar <jar 文件> [args...]
           (执行 jar 文件)
   或  java [options] -m <模块>[/<主类>] [args...]
       java [options] --module <模块>[/<主类>] [args...]
           (执行模块中的主类)
```

```
C:\Users\Administrator
λ where java
D:\Program Files\Java\jdk-10.0.1\bin\java.exe
D:\Program Files\Java\jre-10.0.1\bin\java.exe
C:\Program Files (x86)\Common Files\Oracle\Java\javapath\java.exe
C:\Windows\System32\java.exe

C:\Users\Administrator
λ
```

官方下载地址：Eclipse



选择国内的下载镜像，这样下载速度比较快

下载将会自动开始，若没反应，则可以点击`click here`手动下载

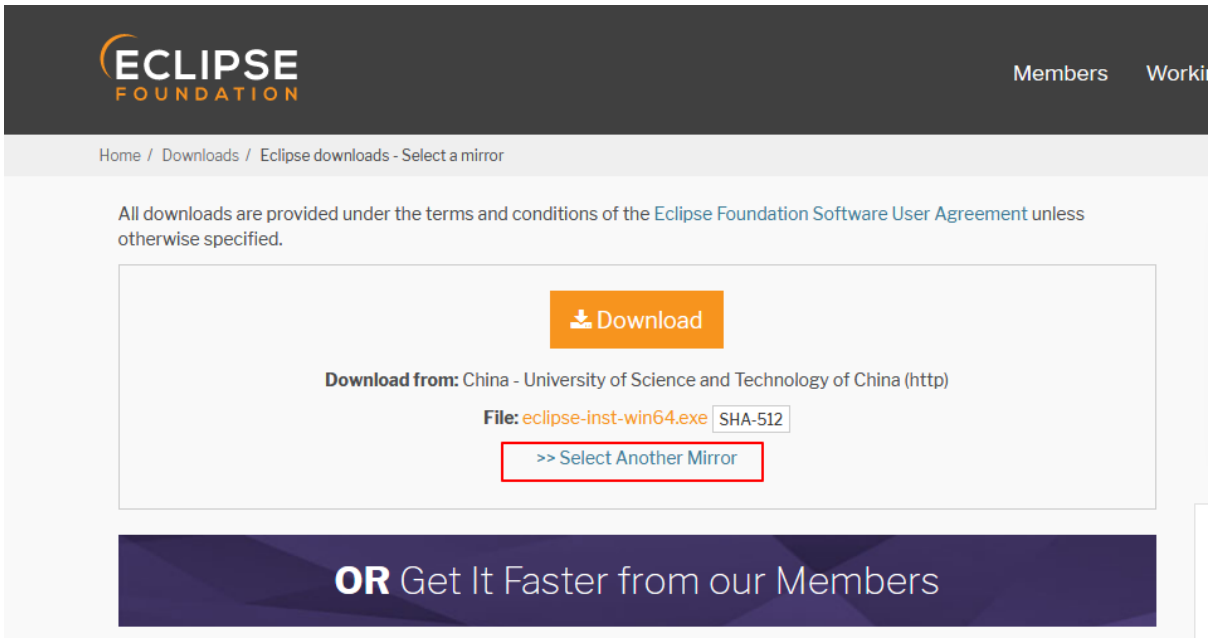
这样下载下来的是一个`Eclipse Installer`安装器，而不是离线安装包，双击它，弹出安装页面，可以选择各种不同的语言的开发环境(包括`Java`、`C/C++`、`JavaEE`、`PHP`等)，这里我选择`C/C++`

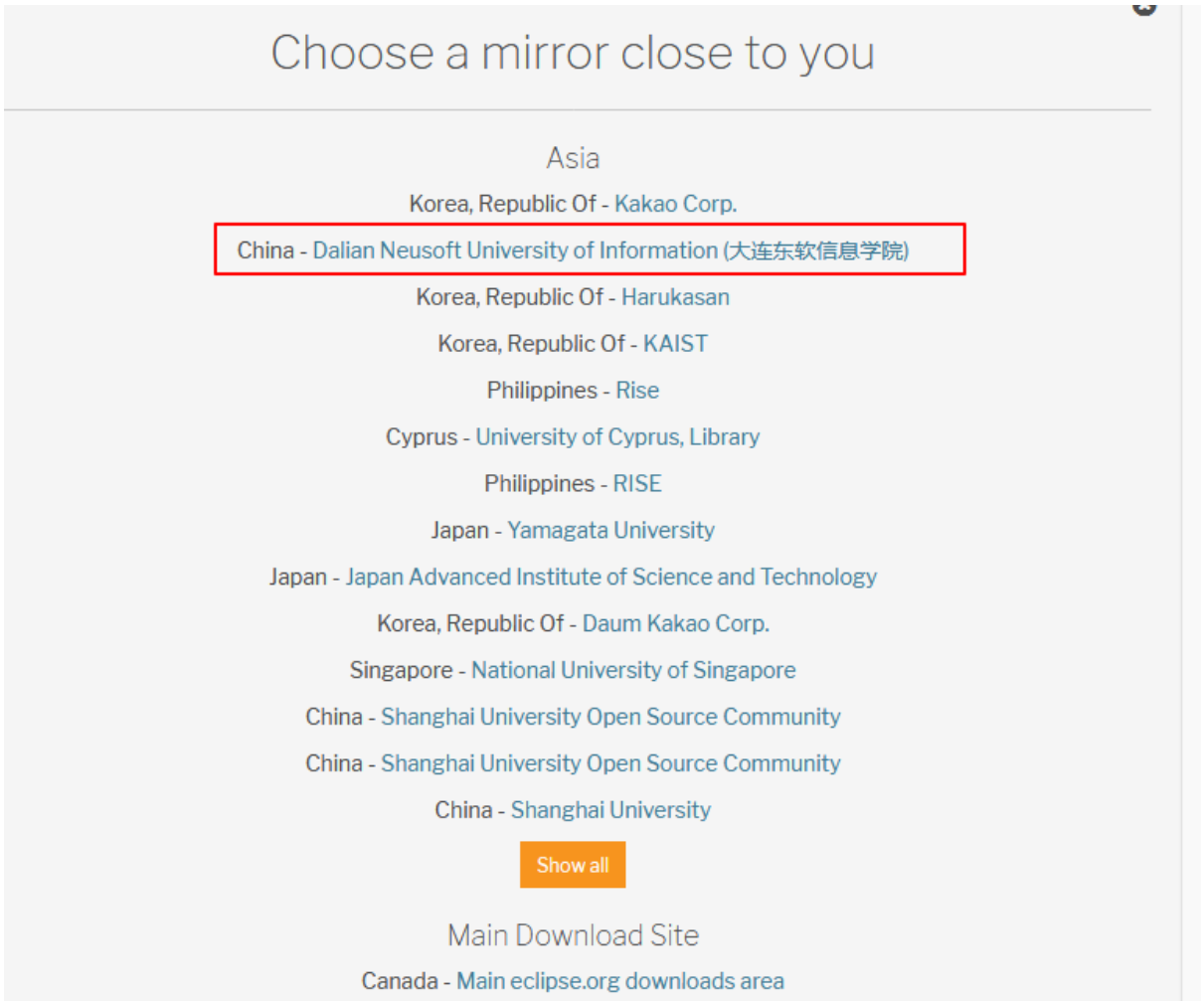
选择安装目录

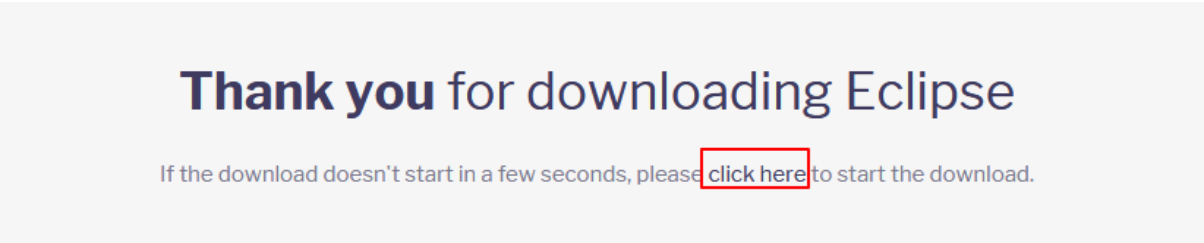
选定安装目录后，点击`INSTALL`即可，接下来我们等待安装完成就可以使用了

基本配置

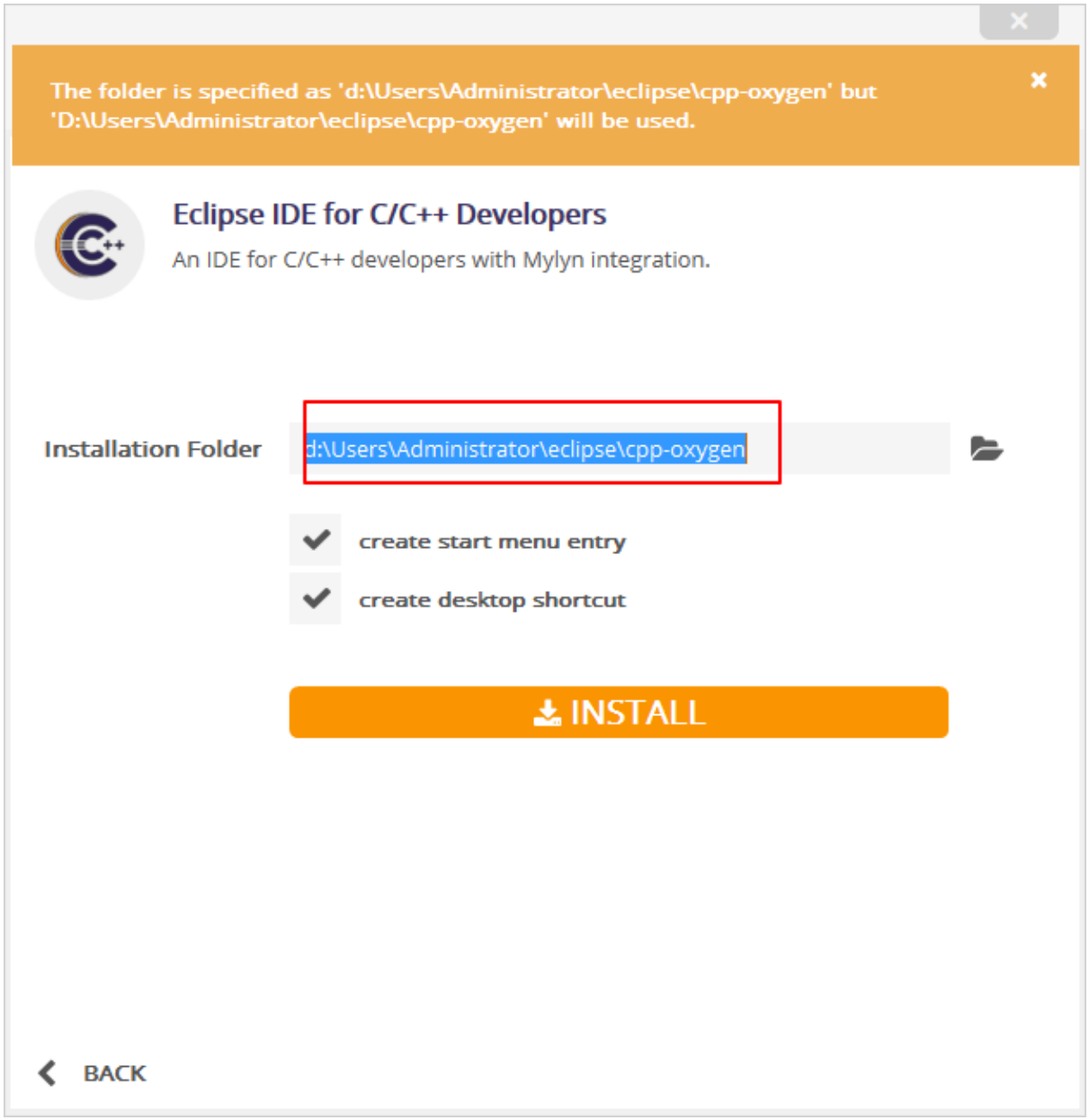
- 映射工作目录
- 配置常用选项

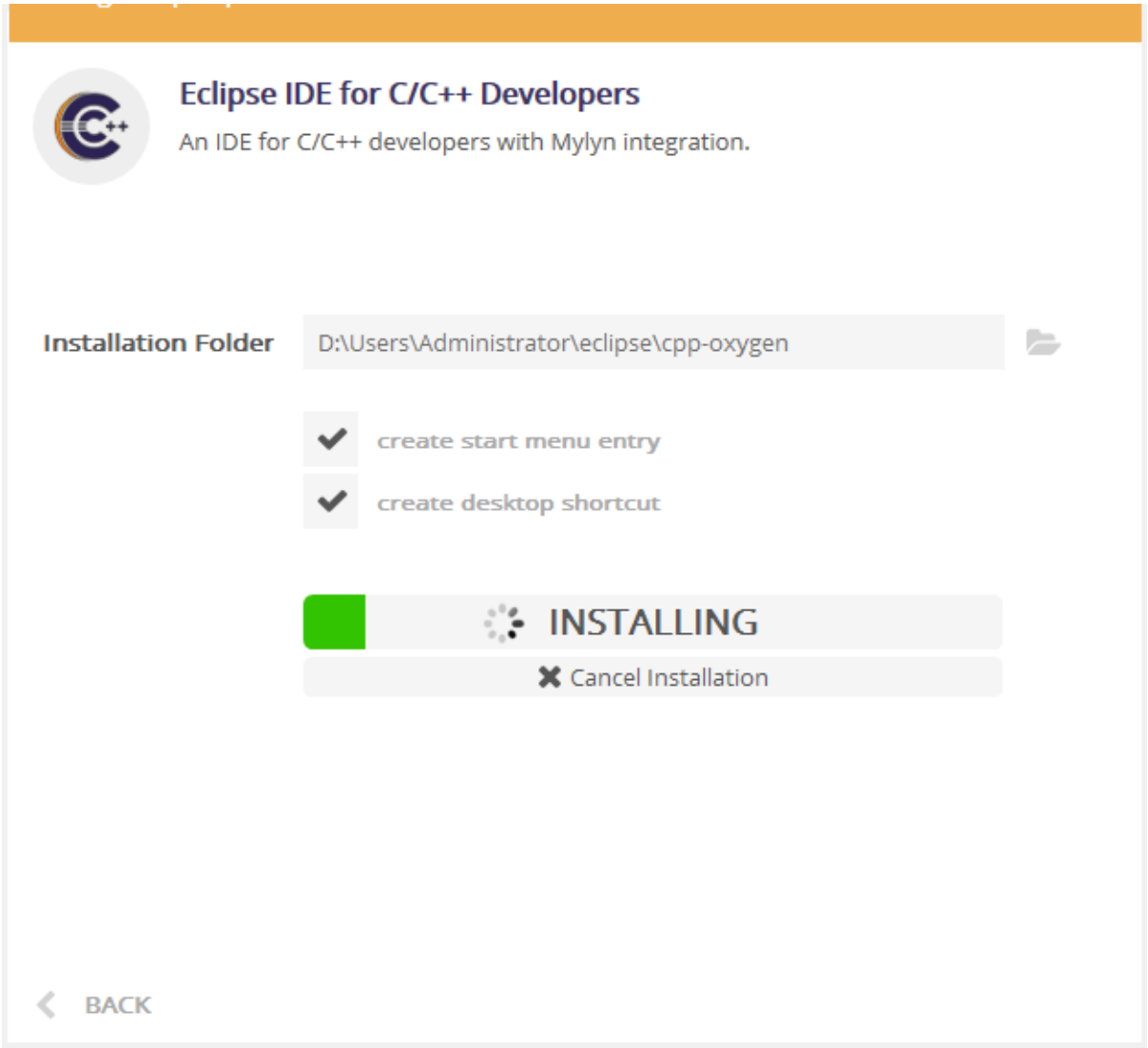

ECLIPSE
FOUNDATION
Members
Worki
Home / Downloads / Eclipse downloads - Select a mirror
All downloads are provided under the terms and conditions of the Eclipse Foundation Software User Agreement unless otherwise specified.
Download
Download from: China - University of Science and Technology of China (http)
File: eclipse-inst-win64.exe SHA-512
>> Select Another Mirror
OR Get It Faster from our Members


Choose a mirror close to you
Asia
Korea, Republic Of - Kakao Corp.
China - Dalian Neusoft University of Information (大连东软信息学院)
Korea, Republic Of - Harukasan
Korea, Republic Of - KAIST
Philippines - Rise
Cyprus - University of Cyprus, Library
Philippines - RISE
Japan - Yamagata University
Japan - Japan Advanced Institute of Science and Technology
Korea, Republic Of - Daum Kakao Corp.
Singapore - National University of Singapore
China - Shanghai University Open Source Community
China - Shanghai University Open Source Community
China - Shanghai University
Show all
Main Download Site
Canada - Main eclipse.org downloads area


Thank you for downloading Eclipse
If the download doesn't start in a few seconds, please click here to start the download.


eclipseinstaller by Oomph
type filter text
Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration
Eclipse IDE for Java EE Developers
Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn, EGit and others.
Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers with Mylyn integration.
Eclipse IDE for JavaScript and Web Developers
The essential tools for any JavaScript developer, including JavaScript, HTML, CSS, XML languages support, Git client, and Mylyn.
Eclipse IDE for PHP Developers
The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, HTML, CSS and XML.


The folder is specified as 'd:\Users\Administrator\eclipse\cpp-oxygen' but
'D:\Users\Administrator\eclipse\cpp-oxygen' will be used.
Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers with Mylyn integration.
Installation Folder
d:\Users\Administrator\eclipse\cpp-oxygen
create start menu entry
create desktop shortcut
INSTALL
BACK

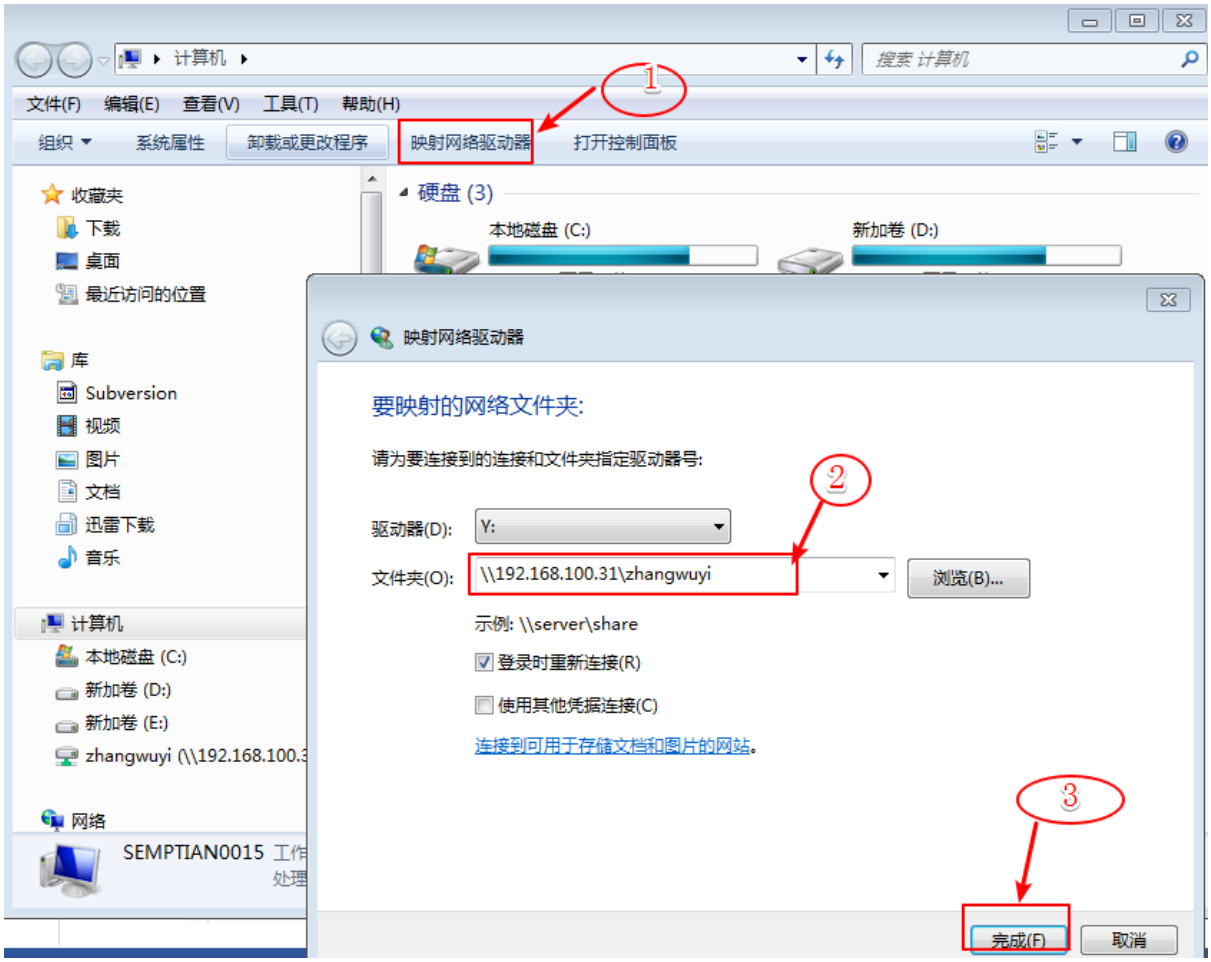

Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers with Mylyn integration.
Installation Folder
D:\Users\Administrator\eclipse\cpp-oxygen
create start menu entry
create desktop shortcut
INSTALLING
Cancel Installation
BACK

- 安装*svn*插件
- 添加*svn*资源库

0x00 映射工作目录

为了方便编译，我们一般采用本地window使用`Eclipse CDT`修改代码，远程linux服务器`gcc`编译的方式

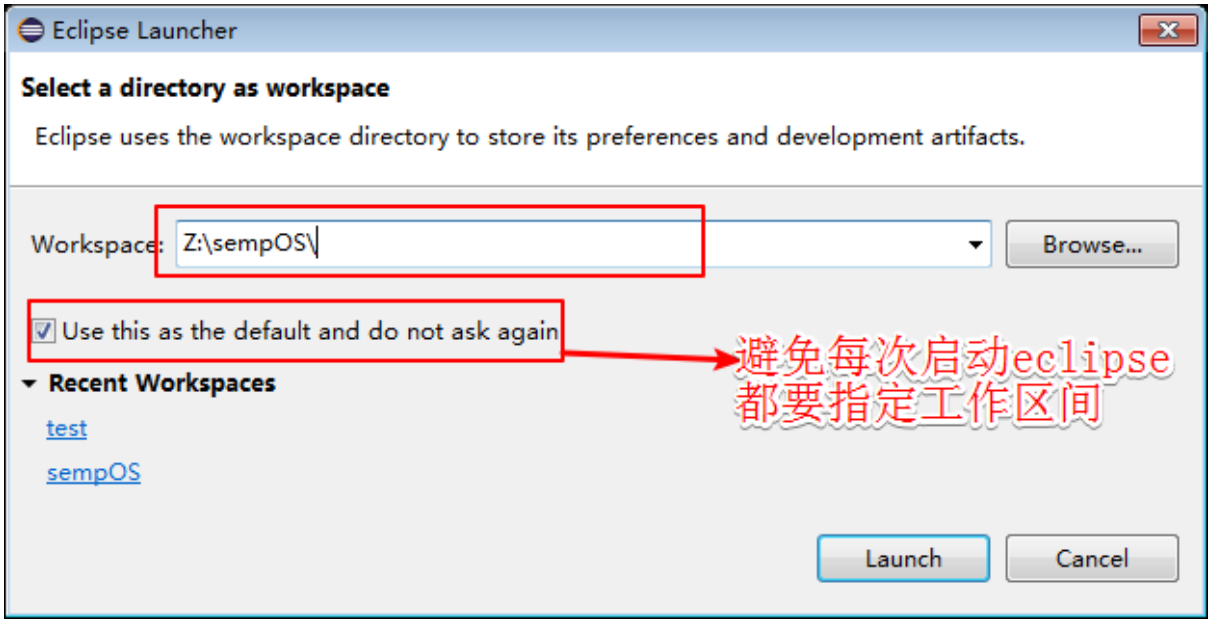
此时就需要将安装`samba`服务的linux服务器上的目录映射到本地进行编辑



0x01 配置常用选项

打开`eclipse`，选择自己默认的工作区间

- **此时将默认工作区间改为刚才映射的工作目录**，因为之后我们检出项目时可以使用默认的工作区间，实现本地和服务器的代码同步
- 可以重新配置默认工作区间，参考Eclipse如何修改eclipse默认的工作空间路径
  - 选择`Window->Preferences->General->Startup and Shutdow->Workspace`
  - 勾选`Prompt for workspaces on startup`
  - 重启`Eclipse`，会和首次启动`Eclipse`一样，弹窗提示设置默认工作区间
- 可以随便切换工作空间：选择`File->Switch Workspace->Other`，可以随时切换到其他工作空间或创建新的工作空间



第一次启动后界面如下，点击右上角的Workbench按钮进入工作空间

点击菜单栏Windows-->Preferences，可以配置相关属性

点击General->Appearance选择喜欢的界面主题

- 个人感觉eclipse的默认主题太丑，安装了个主题插件，可参考：设置漂亮的eclipse主题风格

点击菜单栏Help->Eclipse Marketplace，搜索color eclipse themes，安装eclipse color theme插件就行

安装好之后重启eclipse打开偏好设置General->Preferences, 选择喜欢的theme主题即可

设置字体

设置默认文件编码格式及换行符

设置用空格替换制表符

设置最大行数限制(查看超过该限制的文件时将会关闭部分语法着色功能)

设置常用功能快捷键

0x02 安装svn插件

点击菜单栏Help->Eclipse Marketplace，搜索SVN，安装Subclipse插件

安装完成后重启eclipse

设置SVN接口为SVNKit

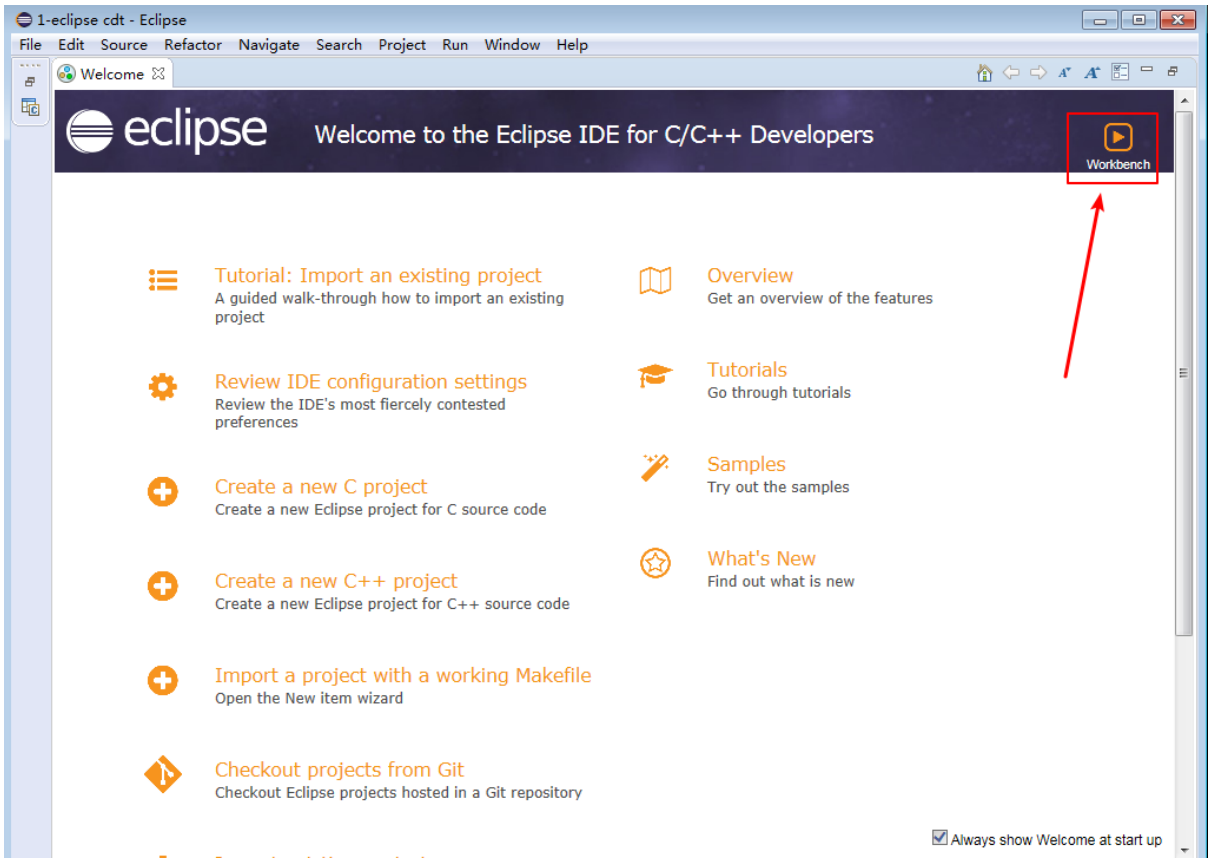
0x03 添加svn资源库

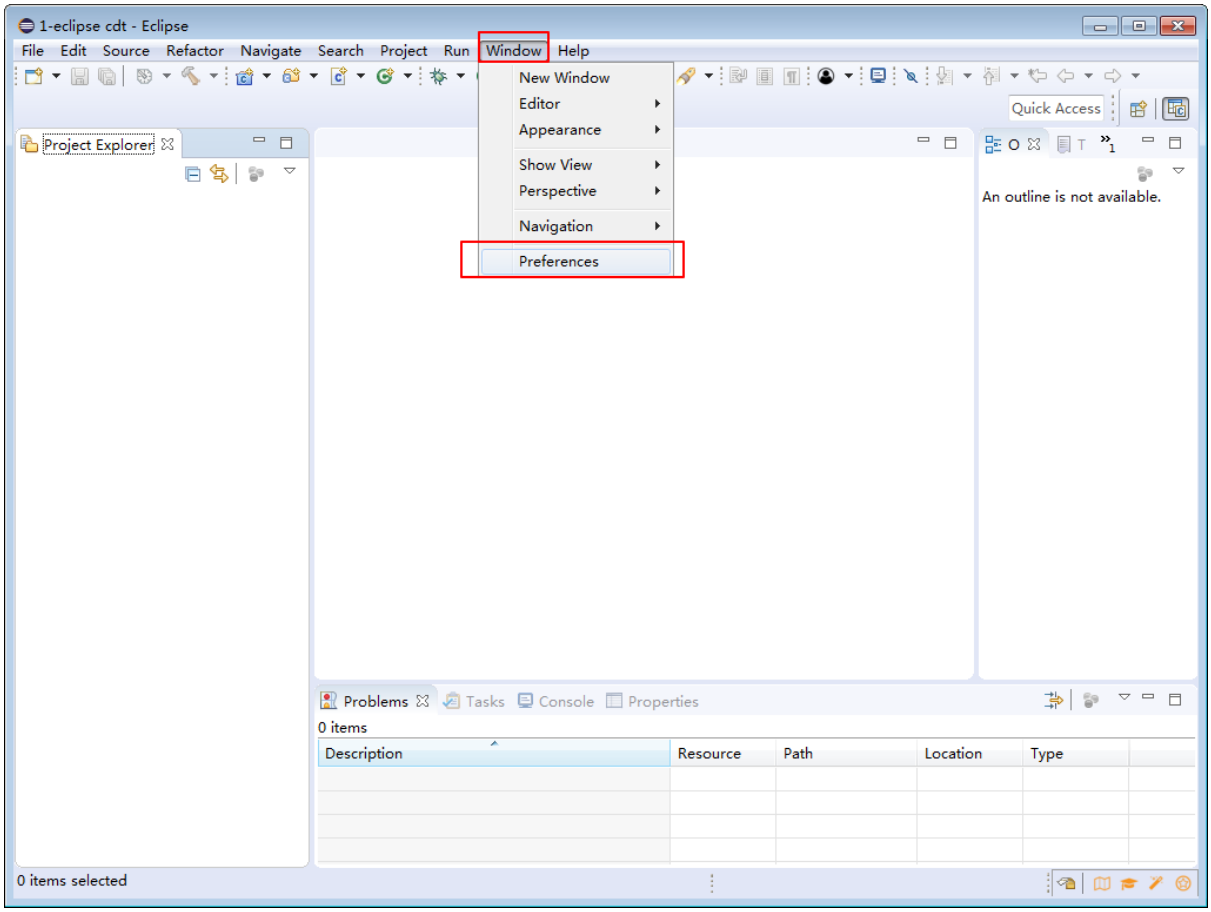
点击右上角Open Perspective按钮，打开SVN资源库视图

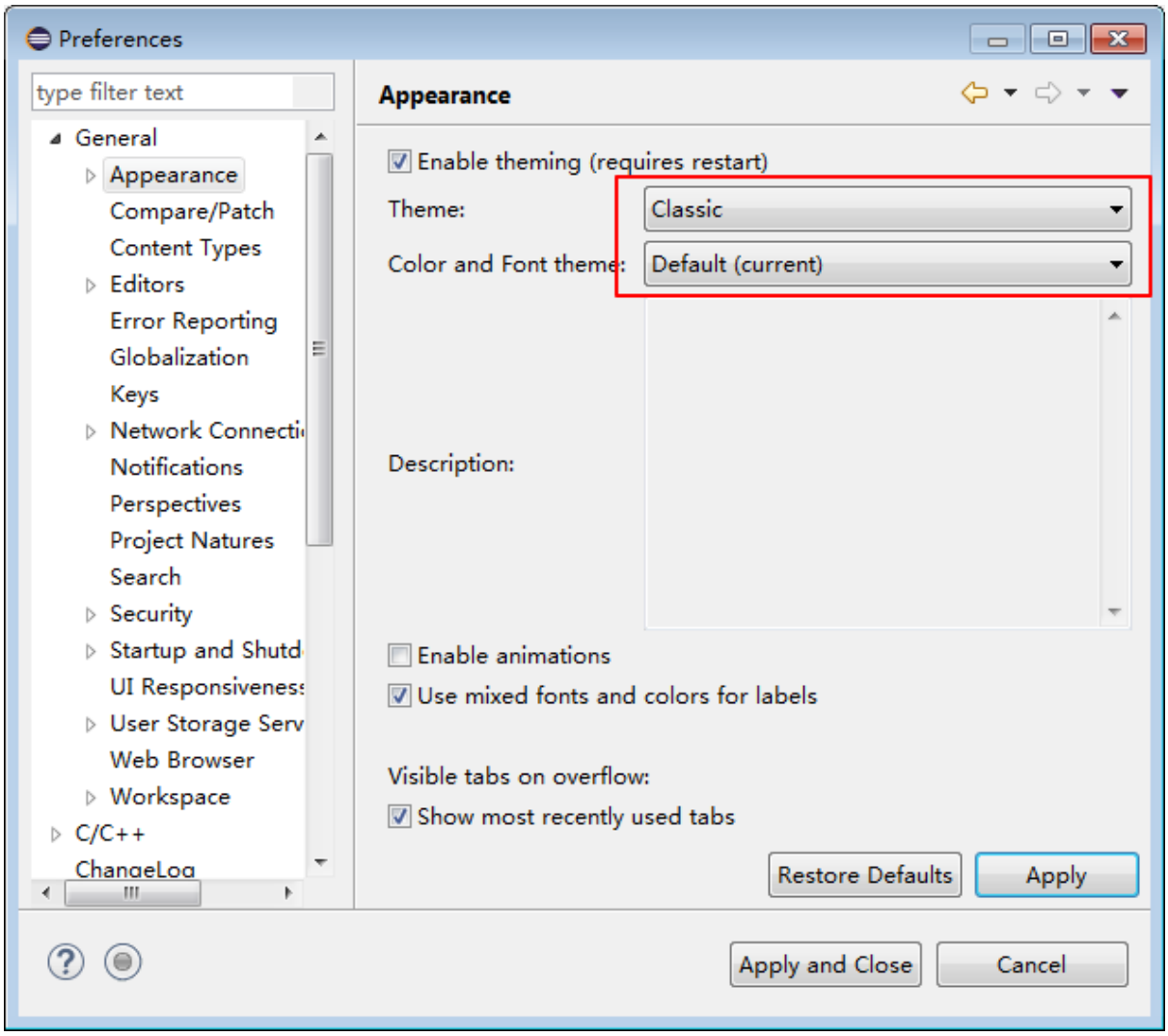
点击svn资源库研究，右键reset 可重置svn资源库，获取相关操作界面

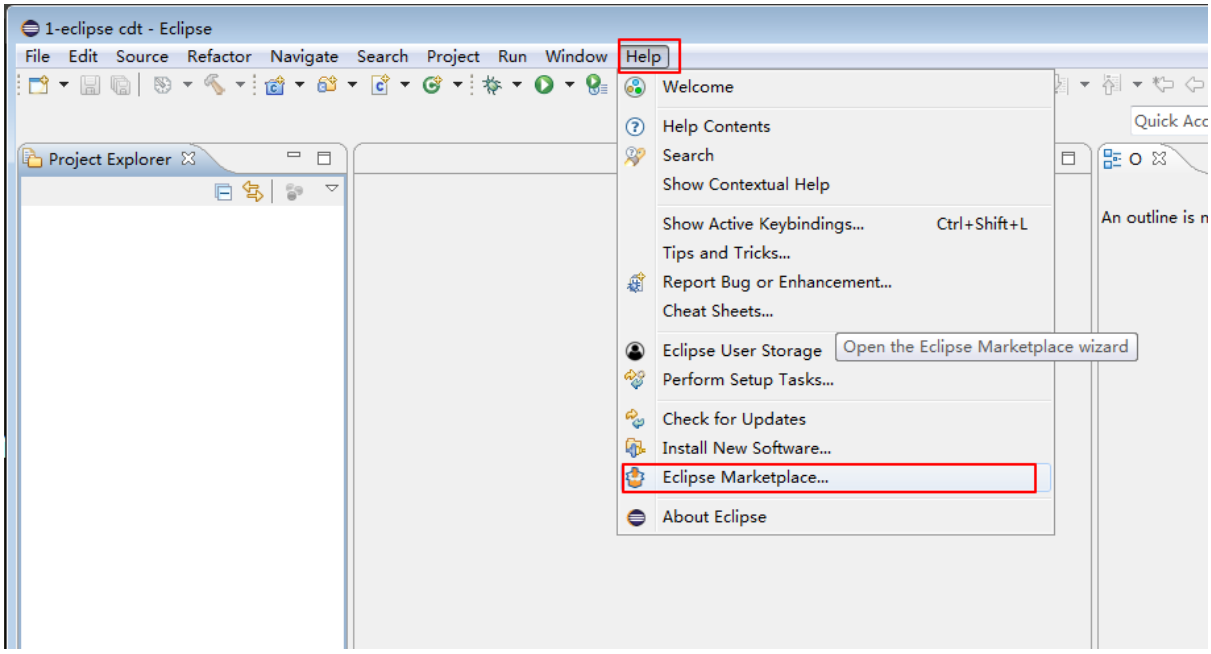
点击添加SVN资源库按钮，输入要添加的代码svn路径

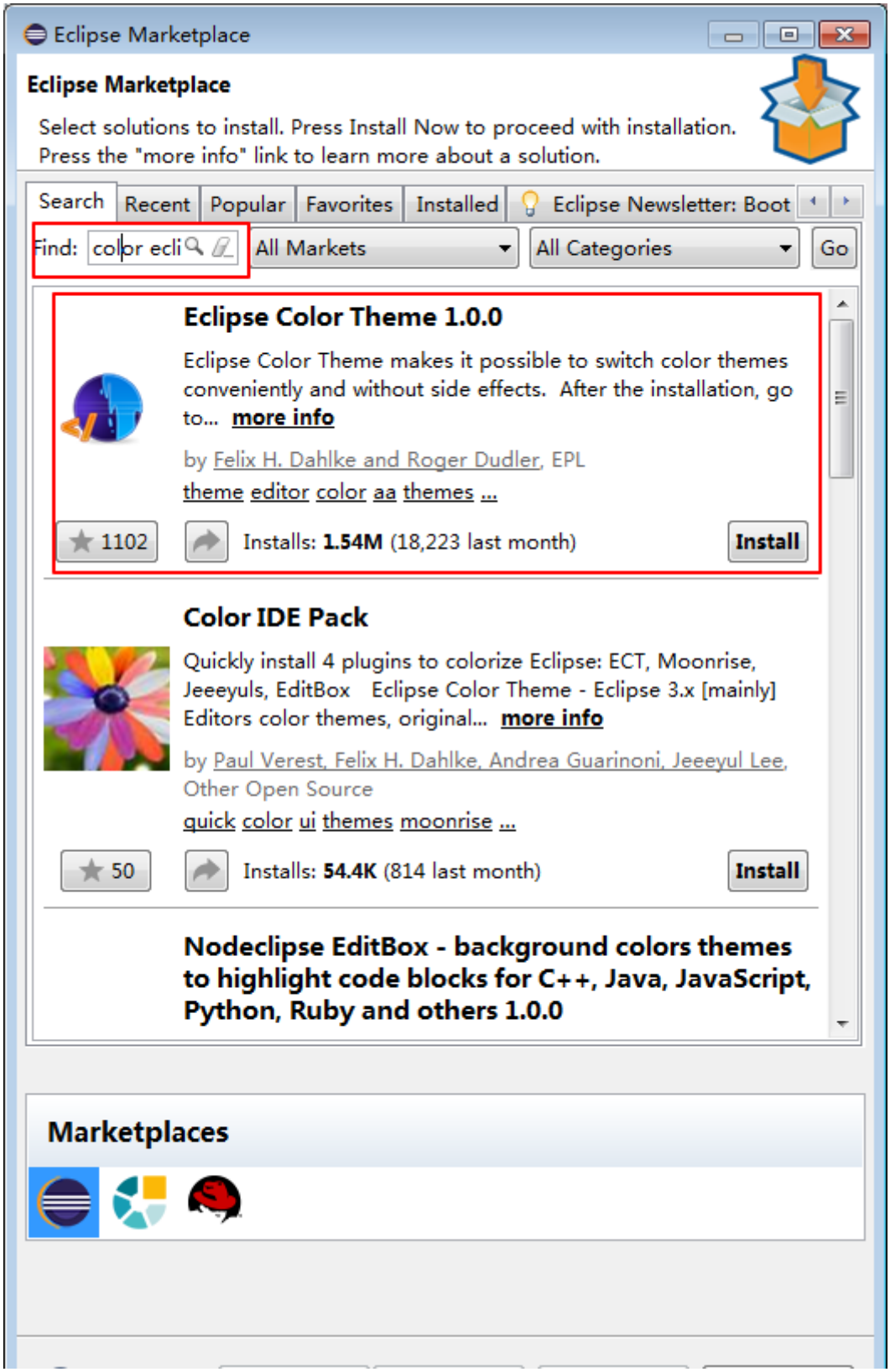
输入SVN账号和密码之后就可以查看已添加的svn资源库的内容

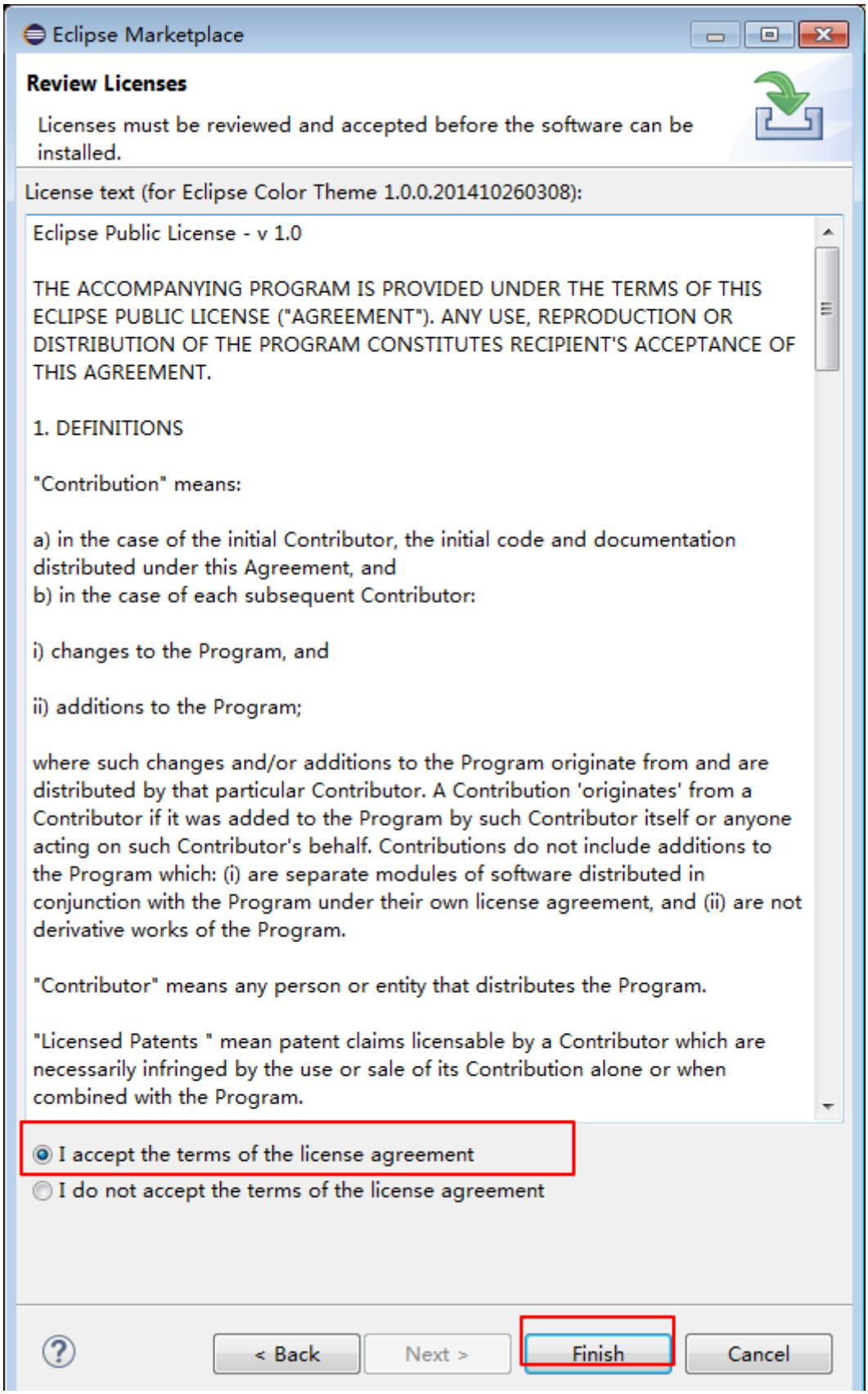

1-eclipse cdt - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Welcome
eclipse
Welcome to the Eclipse IDE for C/C++ Developers
Workbench
Tutorial: Import an existing project
A guided walk-through how to import an existing project
Review IDE configuration settings
Review the IDE's most fiercely contested preferences
Create a new C project
Create a new Eclipse project for C source code
Create a new C++ project
Create a new Eclipse project for C++ source code
Import a project with a working Makefile
Open the New item wizard
Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository
Overview
Get an overview of the features
Tutorials
Go through tutorials
Samples
Try out the samples
What's New
Find out what is new
Always show Welcome at start up

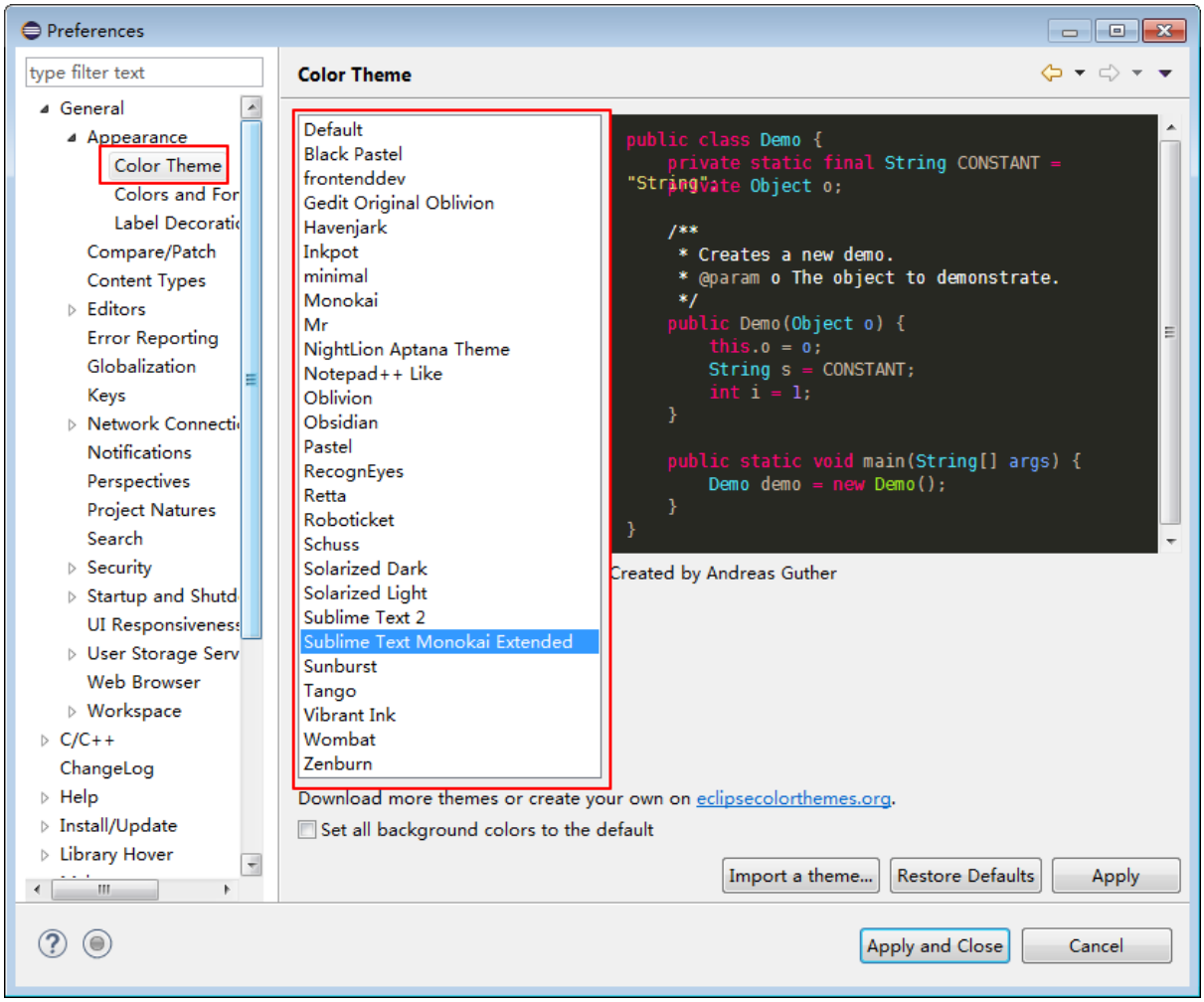

1-eclipse cdt - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
New Window
Editor
Appearance
Show View
Perspective
Navigation
Preferences
Quick Access
Project Explorer
An outline is not available.
Problems
Tasks
Console
Properties
0 items
Description
Resource
Path
Location
Type
0 items selected

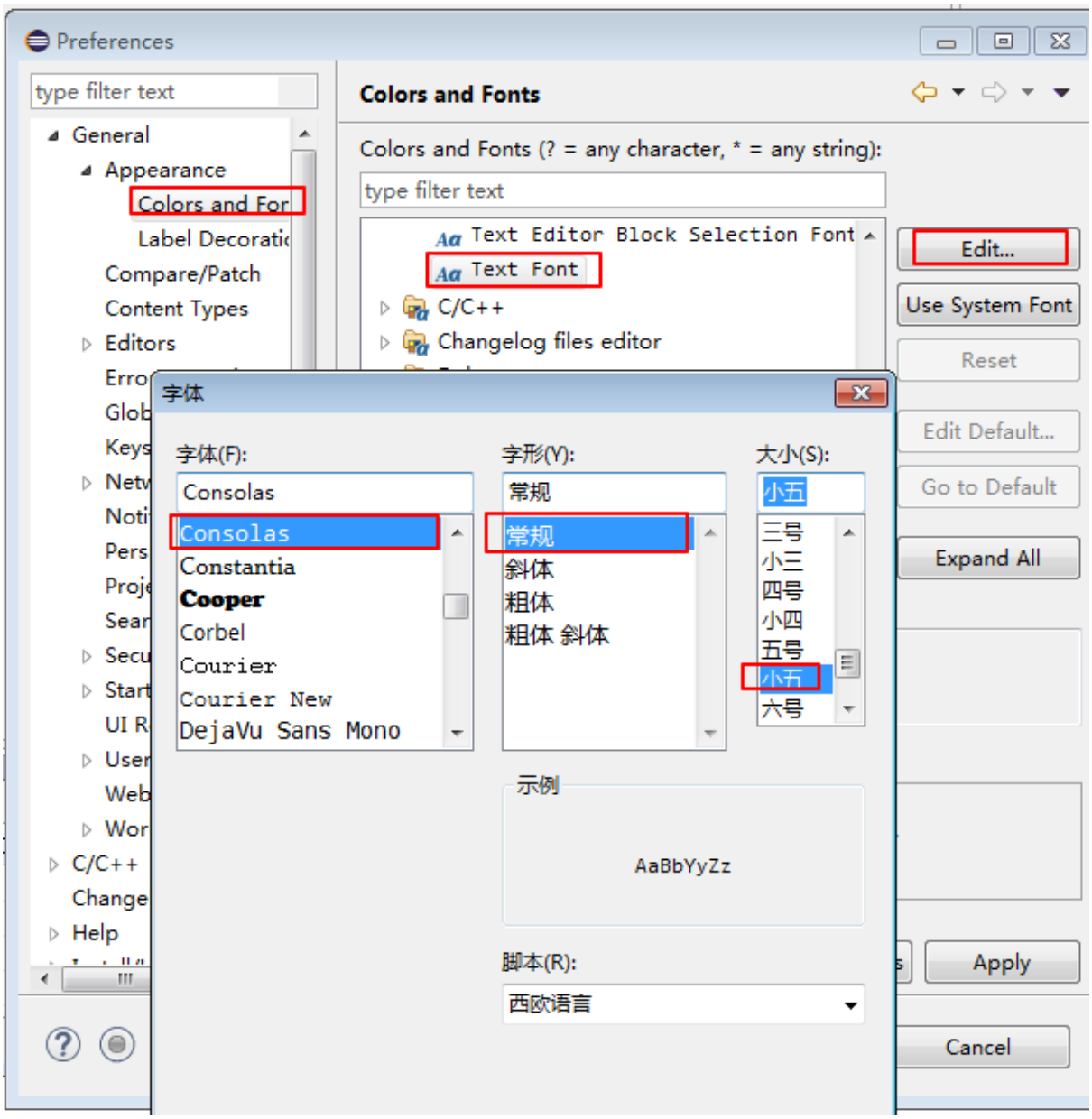

Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Project Natures
Search
Security
Startup and Shutd
UI Responsiveness
User Storage Serv
Web Browser
Workspace
C/C++
ChangeLog
Appearance
Enable theming (requires restart)
Theme:
Classic
Color and Font theme:
Default (current)
Description:
Enable animations
Use mixed fonts and colors for labels
Visible tabs on overflow:
Show most recently used tabs
Restore Defaults
Apply
Apply and Close
Cancel

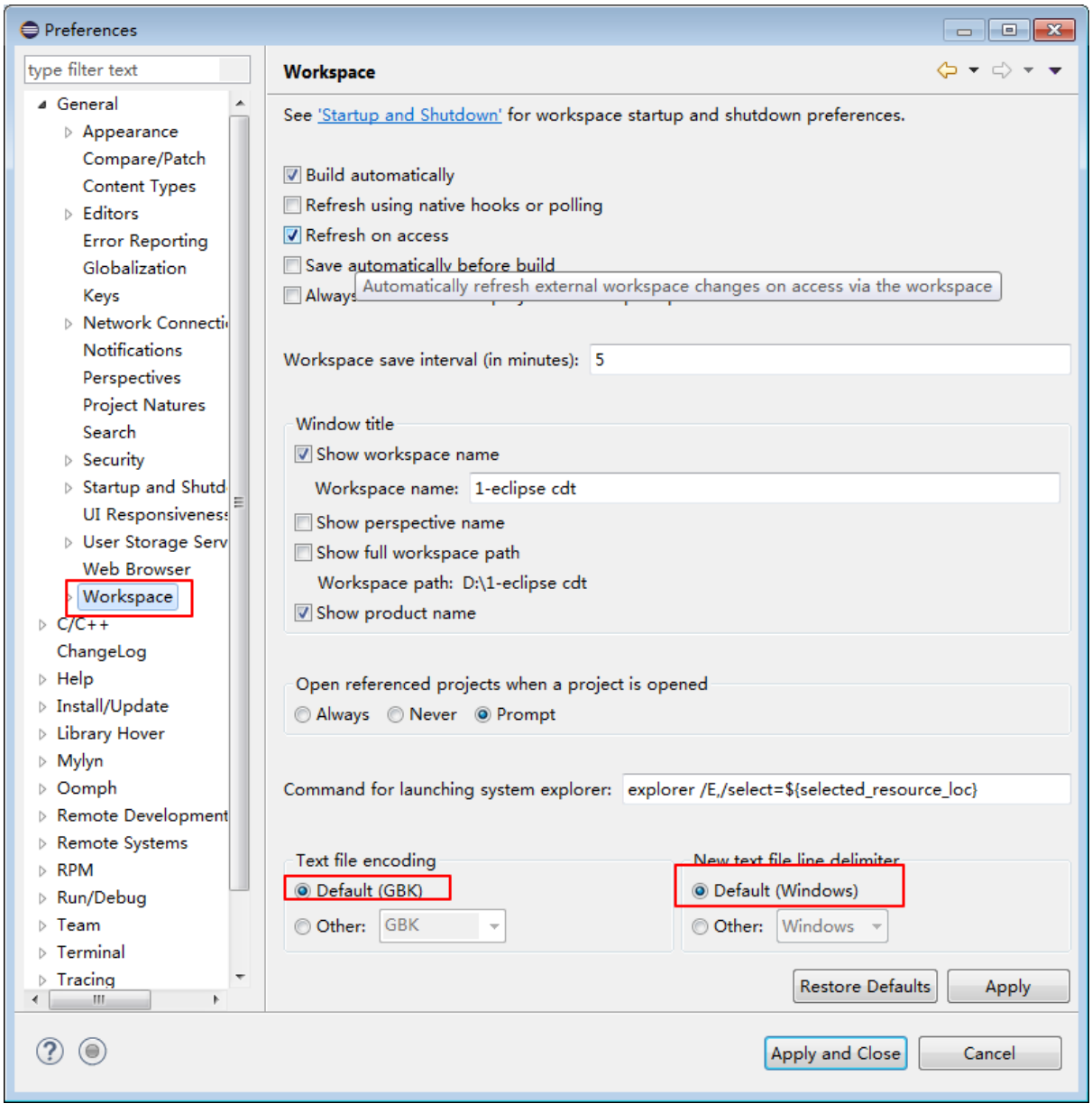

1-eclipse cdt - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Welcome
Help Contents
Search
Show Contextual Help
Show Active Keybindings... Ctrl+Shift+L
Tips and Tricks...
Report Bug or Enhancement...
Cheat Sheets...
Eclipse User Storage
Open the Eclipse Marketplace wizard
Perform Setup Tasks...
Check for Updates
Install New Software...
Eclipse Marketplace...
About Eclipse
An outline is n

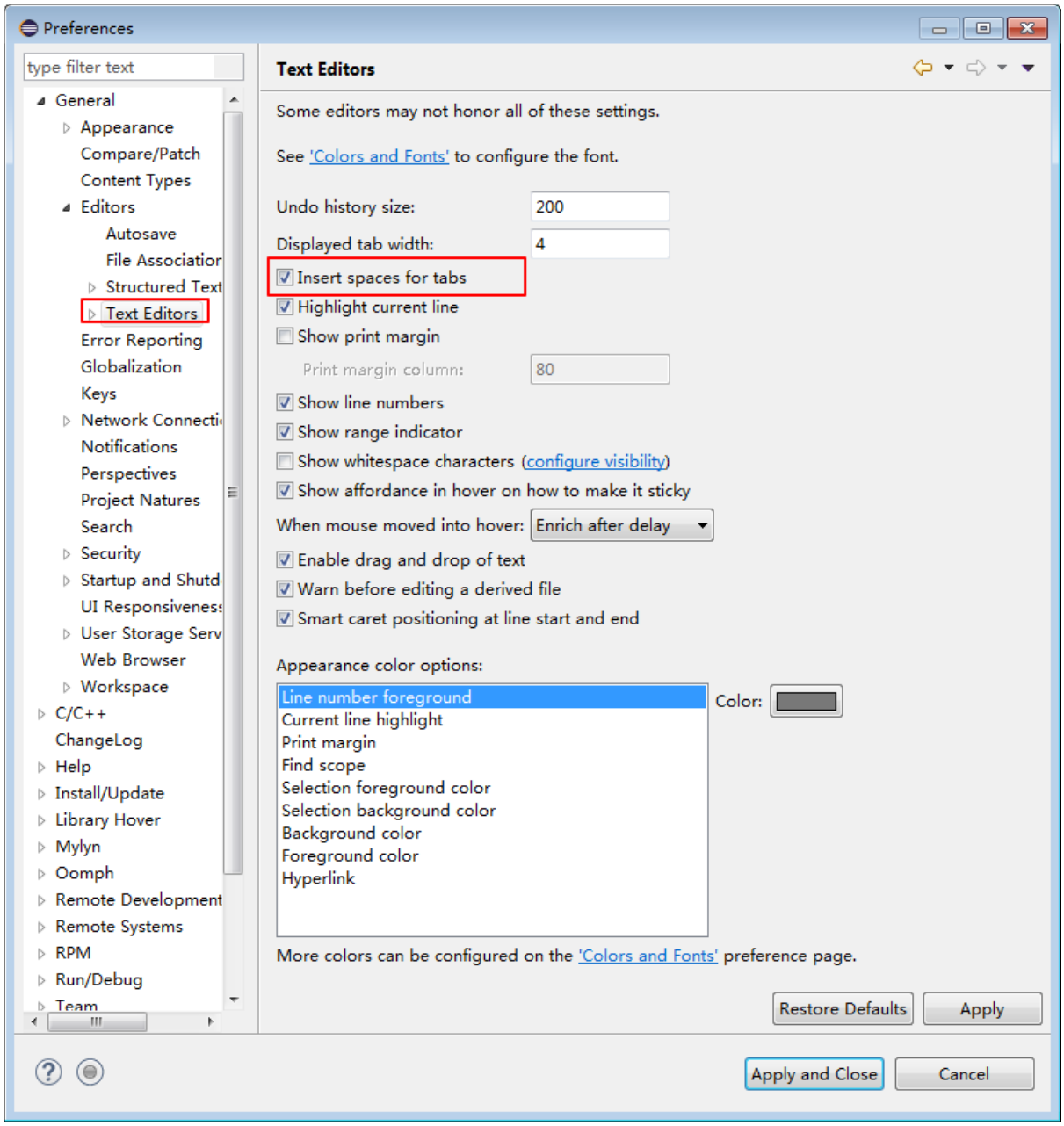

Eclipse Marketplace
Eclipse Marketplace
Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.
Search
Recent
Popular
Favorites
Installed
Eclipse Newsletter: Boot
Find: color ecli
All Markets
All Categories
Go
Eclipse Color Theme 1.0.0
Eclipse Color Theme makes it possible to switch color themes conveniently and without side effects. After the installation, go to... more info
by Felix H. Dahlke and Roger Dudler, EPL
theme editor color aa themes ...
1102
Installs: 1.54M (18,223 last month)
Install
Color IDE Pack
Quickly install 4 plugins to colorize Eclipse: ECT, Moonrise, Jeeeyuls, EditBox Eclipse Color Theme - Eclipse 3.x [mainly] Editors color themes, original... more info
by Paul Verest, Felix H. Dahlke, Andrea Guarinoni, Jeeeyul Lee, Other Open Source
quick color ui themes moonrise ...
50
Installs: 54.4K (814 last month)
Install
Nodeclipse EditBox - background colors themes to highlight code blocks for C++, Java, JavaScript, Python, Ruby and others 1.0.0
Marketplaces

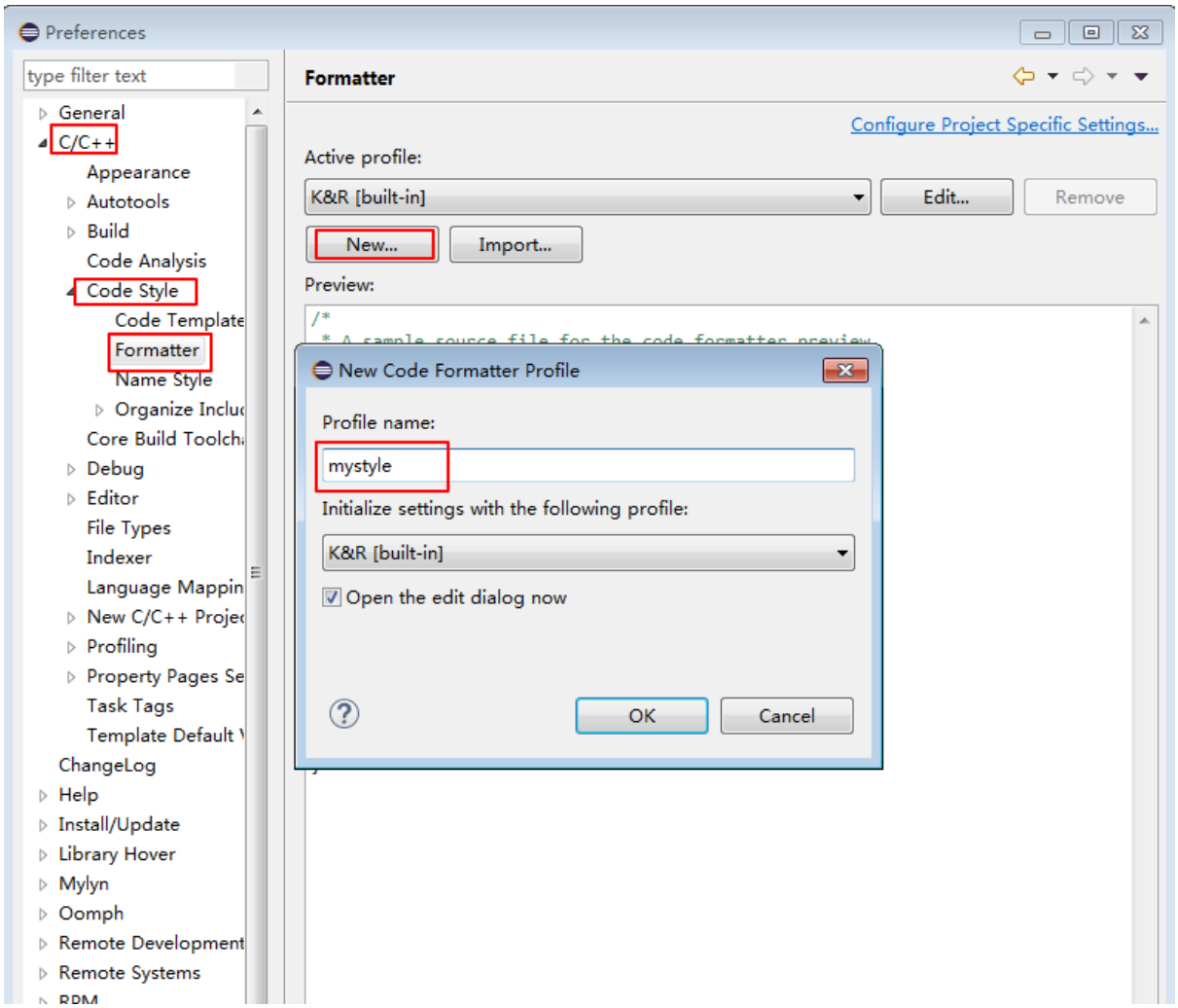

Eclipse Marketplace
Review Licenses
Licenses must be reviewed and accepted before the software can be installed.
License text (for Eclipse Color Theme 1.0.0.201410260308):
Eclipse Public License - v 1.0
THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.
1. DEFINITIONS
"Contribution" means:
a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
b) in the case of each subsequent Contributor:
i) changes to the Program, and
ii) additions to the Program;
where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.
"Contributor" means any person or entity that distributes the Program.
"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.
I accept the terms of the license agreement
I do not accept the terms of the license agreement
< Back
Next >
Finish
Cancel

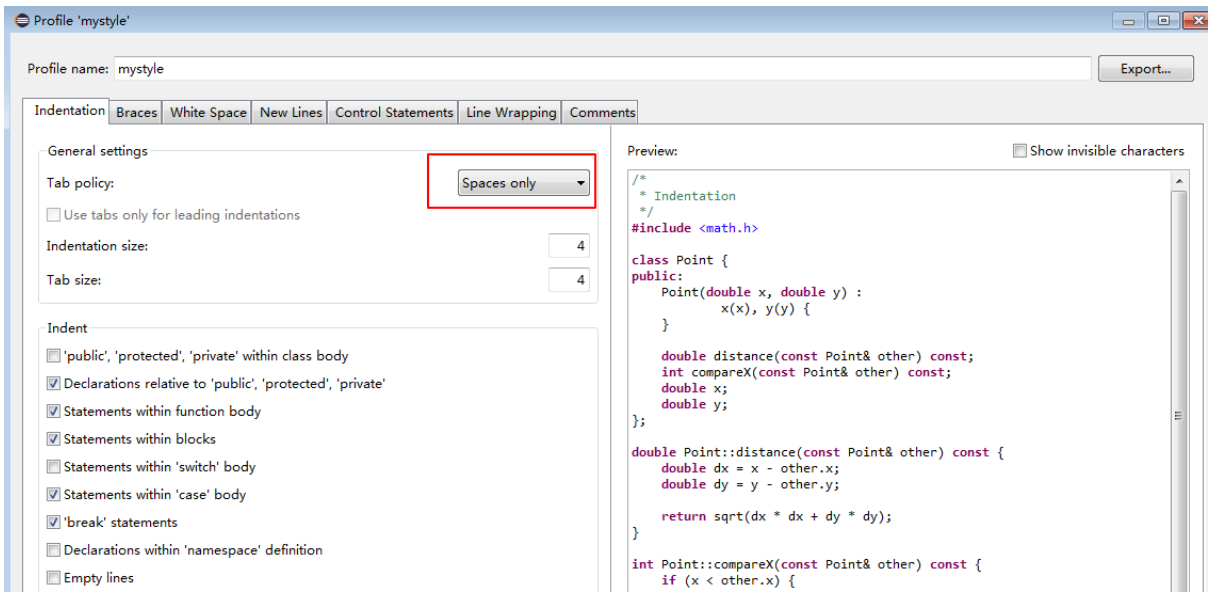

Preferences
type filter text
General
Appearance
Color Theme
Colors and For
Label Decorati
Compare/Patch
Content Types
Editors
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Project Natures
Search
Security
Startup and Shutd
UI Responsiveness
User Storage Serv
Web Browser
Workspace
C/C++
ChangeLog
Help
Install/Update
Library Hover
Color Theme
Default
Black Pastel
frontenddev
Gedit Original Oblivion
Havenjark
Inkpot
minimal
Monokai
Mr
NightLion Aptana Theme
Notepad++ Like
Oblivion
Obsidian
Pastel
RecognEyes
Retta
Roboticket
Schuss
Solarized Dark
Solarized Light
Sublime Text 2
Sublime Text Monokai Extended
Sunburst
Tango
Vibrant Ink
Wombat
Zenburn
public class Demo {
private static final String CONSTANT =
"String";
private Object o;
/**
* Creates a new demo.
* @param o The object to demonstrate.
*/
public Demo(Object o) {
this.o = o;
String s = CONSTANT;
int i = 1;
}
public static void main(String[] args) {
Demo demo = new Demo();
}
}
Created by Andreas Guther
Download more themes or create your own on eclipsecolorthemes.org.
Set all background colors to the default
Import a theme...
Restore Defaults
Apply
Apply and Close
Cancel

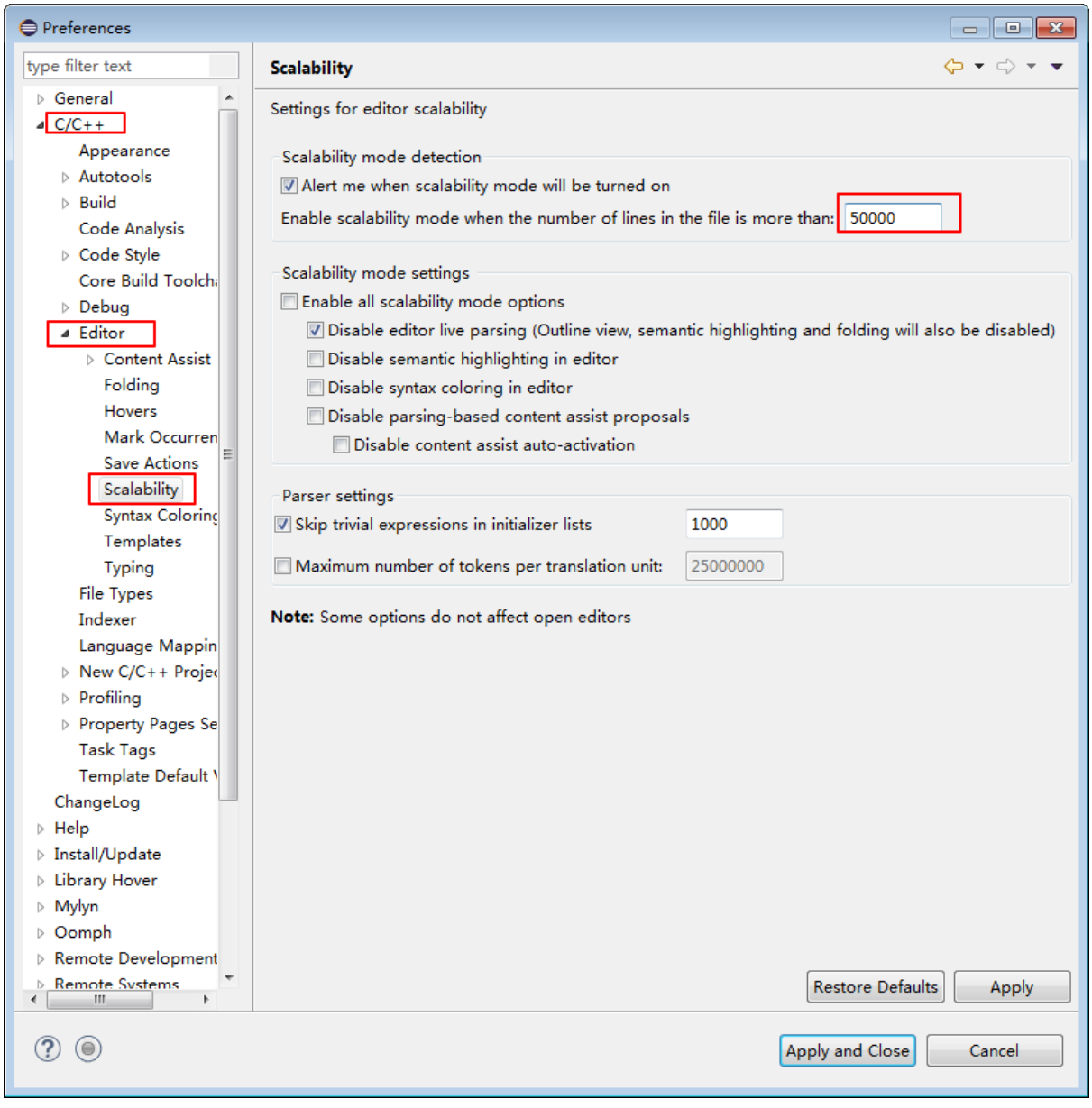

Preferences
type filter text
General
Appearance
Colors and For
Label Decoratic
Compare/Patch
Content Types
Editors
Colors and Fonts
Colors and Fonts (? = any character, * = any string):
type filter text
Text Editor Block Selection Font
Text Font
C/C++
Changelog files editor
Edit...
Use System Font
Reset
Edit Default...
Go to Default
Expand All
Apply
Cancel
字体
字体(F):
字形(Y):
大小(S):
Consolas
常规
小五
Consolas
Constantia
Cooper
Corbel
Courier
Courier New
DejaVu Sans Mono
常规
斜体
粗体
粗体 斜体
三号
小三
四号
小四
五号
小五
六号
示例
AaBbYyZz
脚本(R):
西欧语言

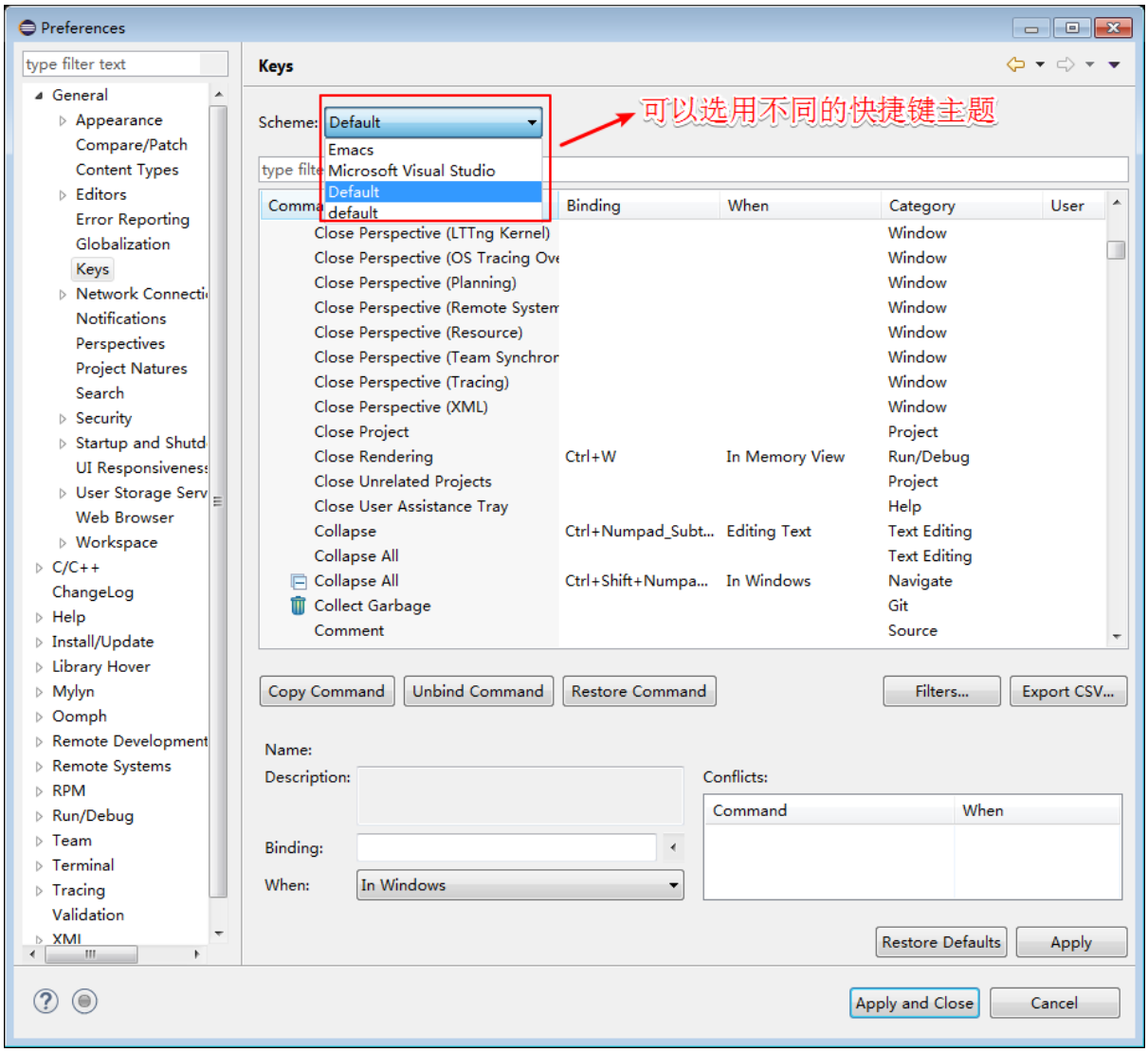

Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Project Natures
Search
Security
Startup and Shutd
UI Responsiveness
User Storage Serv
Web Browser
Workspace
C/C++
ChangeLog
Help
Install/Update
Library Hover
Mylyn
Oomph
Remote Development
Remote Systems
RPM
Run/Debug
Team
Terminal
Tracing
Workspace
See 'Startup and Shutdown' for workspace startup and shutdown preferences.
Build automatically
Refresh using native hooks or polling
Refresh on access
Save automatically before build
Always
Automatically refresh external workspace changes on access via the workspace
Workspace save interval (in minutes): 5
Window title
Show workspace name
Workspace name: 1-eclipse cdt
Show perspective name
Show full workspace path
Workspace path: D:\1-eclipse cdt
Show product name
Open referenced projects when a project is opened
Always
Never
Prompt
Command for launching system explorer: explorer /E,/select=${selected_resource_loc}
Text file encoding
Default (GBK)
Other: GBK
New text file line delimiter
Default (Windows)
Other: Windows
Restore Defaults
Apply
Apply and Close
Cancel

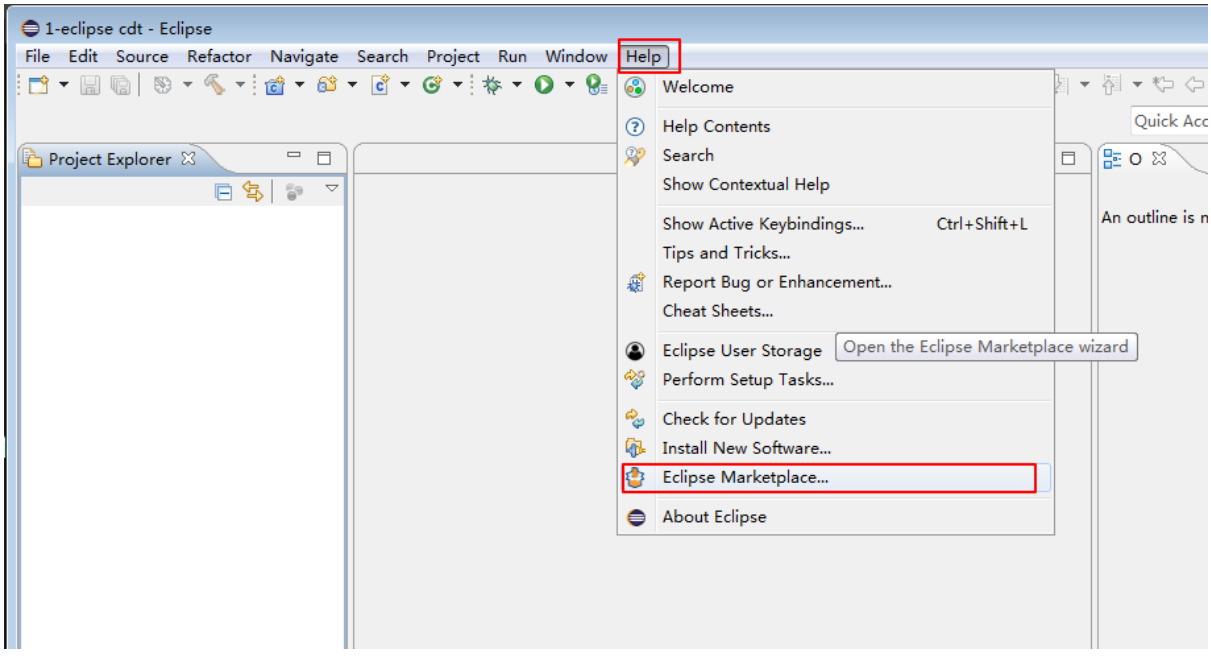

Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Autosave
File Association
Structured Text
Text Editors
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Project Natures
Search
Security
Startup and Shutd
UI Responsiveness
User Storage Serv
Web Browser
Workspace
C/C++
ChangeLog
Help
Install/Update
Library Hover
Mylyn
Oomph
Remote Development
Remote Systems
RPM
Run/Debug
Team
Text Editors
Some editors may not honor all of these settings.
See 'Colors and Fonts' to configure the font.
Undo history size: 200
Displayed tab width: 4
Insert spaces for tabs
Highlight current line
Show print margin
Print margin column: 80
Show line numbers
Show range indicator
Show whitespace characters (configure visibility)
Show affordance in hover on how to make it sticky
When mouse moved into hover: Enrich after delay
Enable drag and drop of text
Warn before editing a derived file
Smart caret positioning at line start and end
Appearance color options:
Line number foreground
Current line highlight
Print margin
Find scope
Selection foreground color
Selection background color
Background color
Foreground color
Hyperlink
Color:
More colors can be configured on the 'Colors and Fonts' preference page.
Restore Defaults
Apply
Apply and Close
Cancel

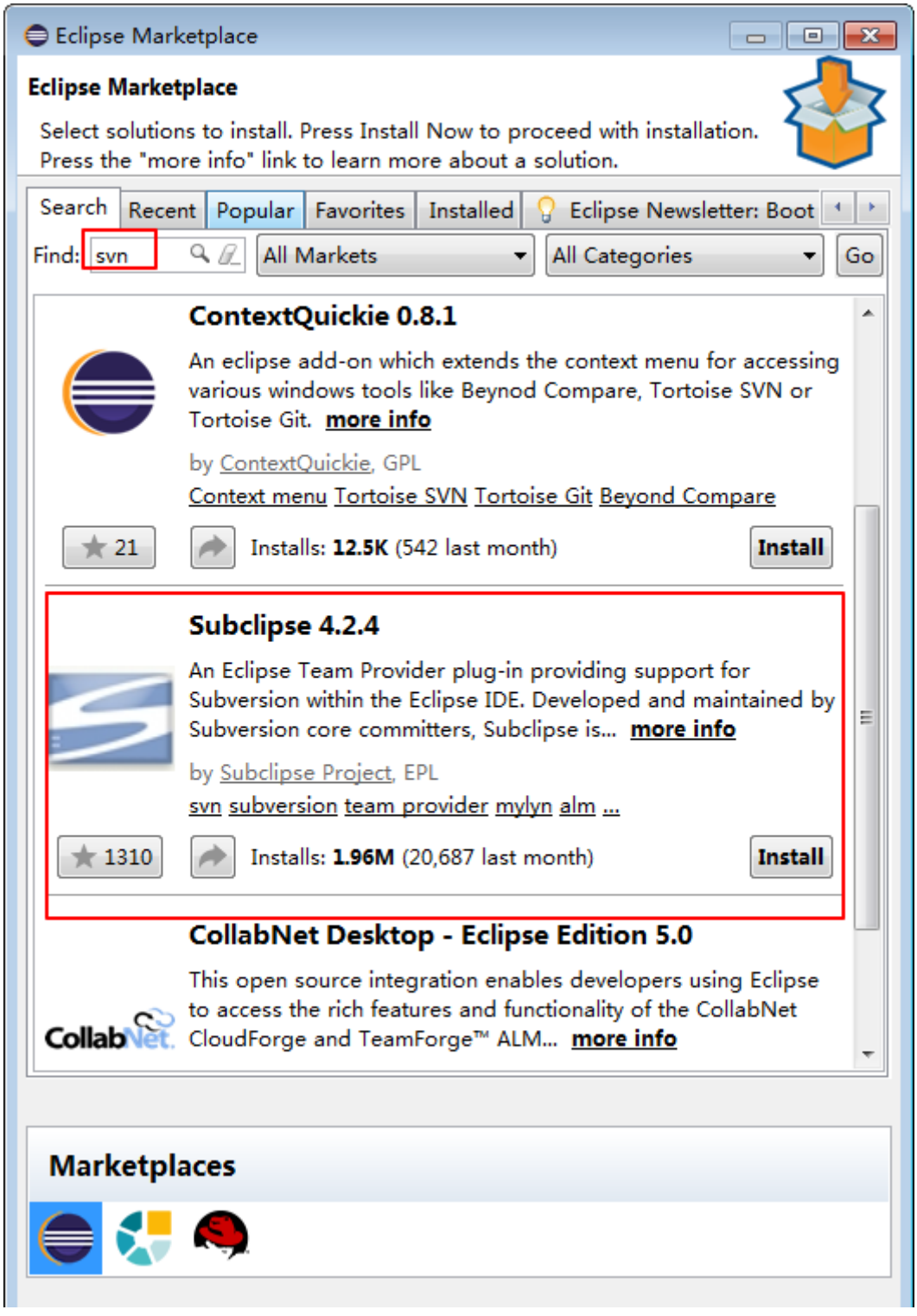

Preferences
type filter text
General
C/C++
Appearance
Autotools
Build
Code Analysis
Code Style
Code Template
Formatter
Name Style
Organize Inclu
Core Build Toolch
Debug
Editor
File Types
Indexer
Language Mappin
New C/C++ Projec
Profiling
Property Pages Se
Task Tags
Template Default
ChangeLog
Help
Install/Update
Library Hover
Mylyn
Oomph
Remote Development
Remote Systems
Formatter
Configure Project Specific Settings...
Active profile:
K&R [built-in]
Edit...
Remove
New...
Import...
Preview:
New Code Formatter Profile
Profile name:
mystyle
Initialize settings with the following profile:
K&R [built-in]
Open the edit dialog now
OK
Cancel

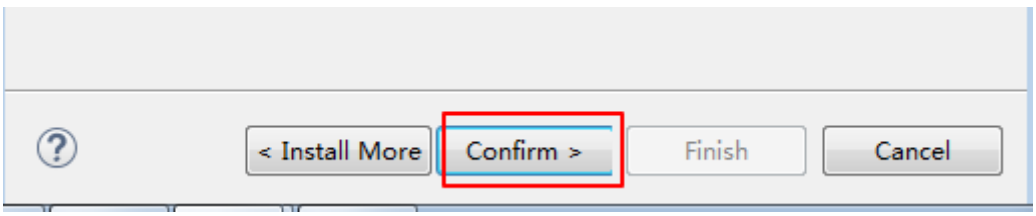

Profile 'mystyle'
Profile name: mystyle
Export...
Indentation
Braces
White Space
New Lines
Control Statements
Line Wrapping
Comments
General settings
Tab policy:
Spaces only
Use tabs only for leading indentations
Indentation size: 4
Tab size: 4
Indent
'public', 'protected', 'private' within class body
Declarations relative to 'public', 'protected', 'private'
Statements within function body
Statements within blocks
Statements within 'switch' body
Statements within 'case' body
'break' statements
Declarations within 'namespace' definition
Empty lines
Preview:
Show invisible characters
/*
* Indentation
*/
#include <math.h>
class Point {
public:
Point(double x, double y) :
x(x), y(y) {
}
double distance(const Point& other) const;
int compareX(const Point& other) const;
double x;
double y;
};
double Point::distance(const Point& other) const {
double dx = x - other.x;
double dy = y - other.y;
return sqrt(dx * dx + dy * dy);
}
int Point::compareX(const Point& other) const {
if (x < other.x) {

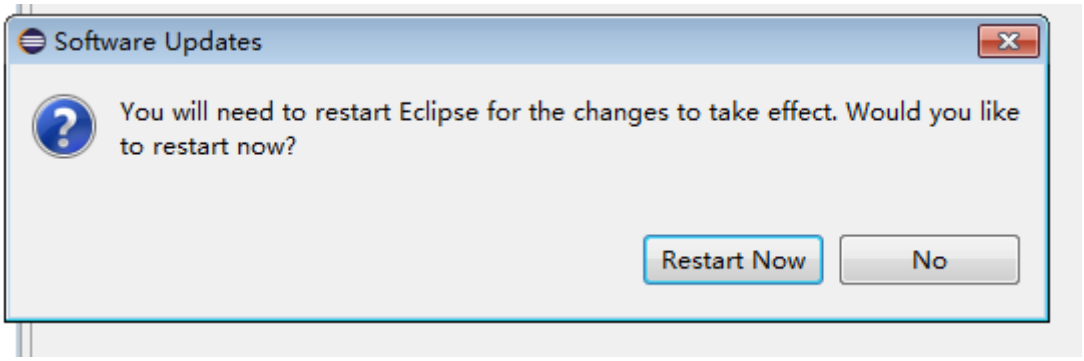

Preferences
type filter text
General
C/C++
Appearance
Autotools
Build
Code Analysis
Code Style
Core Build Toolch
Debug
Editor
Content Assist
Folding
Hovers
Mark Occurren
Save Actions
Scalability
Syntax Coloring
Templates
Typing
File Types
Indexer
Language Mappin
New C/C++ Proje
Profiling
Property Pages Se
Task Tags
Template Default
ChangeLog
Help
Install/Update
Library Hover
Mylyn
Oomph
Remote Development
Remote Systems
Scalability
Settings for editor scalability
Scalability mode detection
Alert me when scalability mode will be turned on
Enable scalability mode when the number of lines in the file is more than: 50000
Scalability mode settings
Enable all scalability mode options
Disable editor live parsing (Outline view, semantic highlighting and folding will also be disabled)
Disable semantic highlighting in editor
Disable syntax coloring in editor
Disable parsing-based content assist proposals
Disable content assist auto-activation
Parser settings
Skip trivial expressions in initializer lists 1000
Maximum number of tokens per translation unit: 25000000
Note: Some options do not affect open editors
Restore Defaults
Apply
Apply and Close
Cancel


Preferences
type filter text
General
Appearance
Compare/Patch
Content Types
Editors
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Project Natures
Search
Security
Startup and Shutd
UI Responsiveness
User Storage Serv
Web Browser
Workspace
C/C++
ChangeLog
Help
Install/Update
Library Hover
Mylyn
Oomph
Remote Development
Remote Systems
RPM
Run/Debug
Team
Terminal
Tracing
Validation
XML
Keys
Scheme:
Default
Emacs
Microsoft Visual Studio
Default
default
可以选用不同的快捷键主题
Binding
When
Category
User
Close Perspective (LTTng Kernel)
Close Perspective (OS Tracing Ove
Close Perspective (Planning)
Close Perspective (Remote Systen
Close Perspective (Resource)
Close Perspective (Team Synchror
Close Perspective (Tracing)
Close Perspective (XML)
Close Project
Close Rendering
Close Unrelated Projects
Close User Assistance Tray
Collapse
Collapse All
Collapse All
Collect Garbage
Comment
Ctrl+W
In Memory View
Ctrl+Numpad_Subt...
Editing Text
Ctrl+Shift+Numpa...
In Windows
Window
Project
Run/Debug
Help
Text Editing
Text Editing
Navigate
Git
Source
Copy Command
Unbind Command
Restore Command
Filters...
Export CSV...
Name:
Description:
Binding:
When:
In Windows
Conflicts:
Command
When
Restore Defaults
Apply
Apply and Close
Cancel


1-eclipse cdt - Eclipse
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
Project Explorer
Welcome
Help Contents
Search
Show Contextual Help
Show Active Keybindings...
Ctrl+Shift+L
Tips and Tricks...
Report Bug or Enhancement...
Cheat Sheets...
Eclipse User Storage
Open the Eclipse Marketplace wizard
Perform Setup Tasks...
Check for Updates
Install New Software...
Eclipse Marketplace...
About Eclipse
An outline is n


Eclipse Marketplace
Eclipse Marketplace
Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.
Search
Recent
Popular
Favorites
Installed
Eclipse Newsletter: Boot
Find:
svn
All Markets
All Categories
Go
ContextQuickie 0.8.1
An eclipse add-on which extends the context menu for accessing various windows tools like Beynod Compare, Tortoise SVN or Tortoise Git. more info
by ContextQuickie, GPL
Context menu Tortoise SVN Tortoise Git Beyond Compare
21
Installs: 12.5K (542 last month)
Install
Subclipse 4.2.4
An Eclipse Team Provider plug-in providing support for Subversion within the Eclipse IDE. Developed and maintained by Subversion core committers, Subclipse is... more info
by Subclipse Project, EPL
svn subversion team provider mylyn alm ...
1310
Installs: 1.96M (20,687 last month)
Install
CollabNet Desktop - Eclipse Edition 5.0
This open source integration enables developers using Eclipse to access the rich features and functionality of the CollabNet CloudForge and TeamForge™ ALM... more info
CollabNet
Marketplaces


< Install More
Confirm >
Finish
Cancel



项目工程管理

- 创建c项目
  - *svn*仓库检出创建
  - *eclipse*手动新建
- 设置项目选项
- 代码编辑
- 代码编译
- 代码提交

0x00 创建c项目

创建c项目的方法有两种:

- eclipse导入svn资源库全部源码，从svn资源库检出指定目录创建项目(**创建速度较慢，不推荐**)
- svn下载全部源码到服务器，然后eclipse指定路径手动创建项目(**创建速度较快，推荐此种方法**)

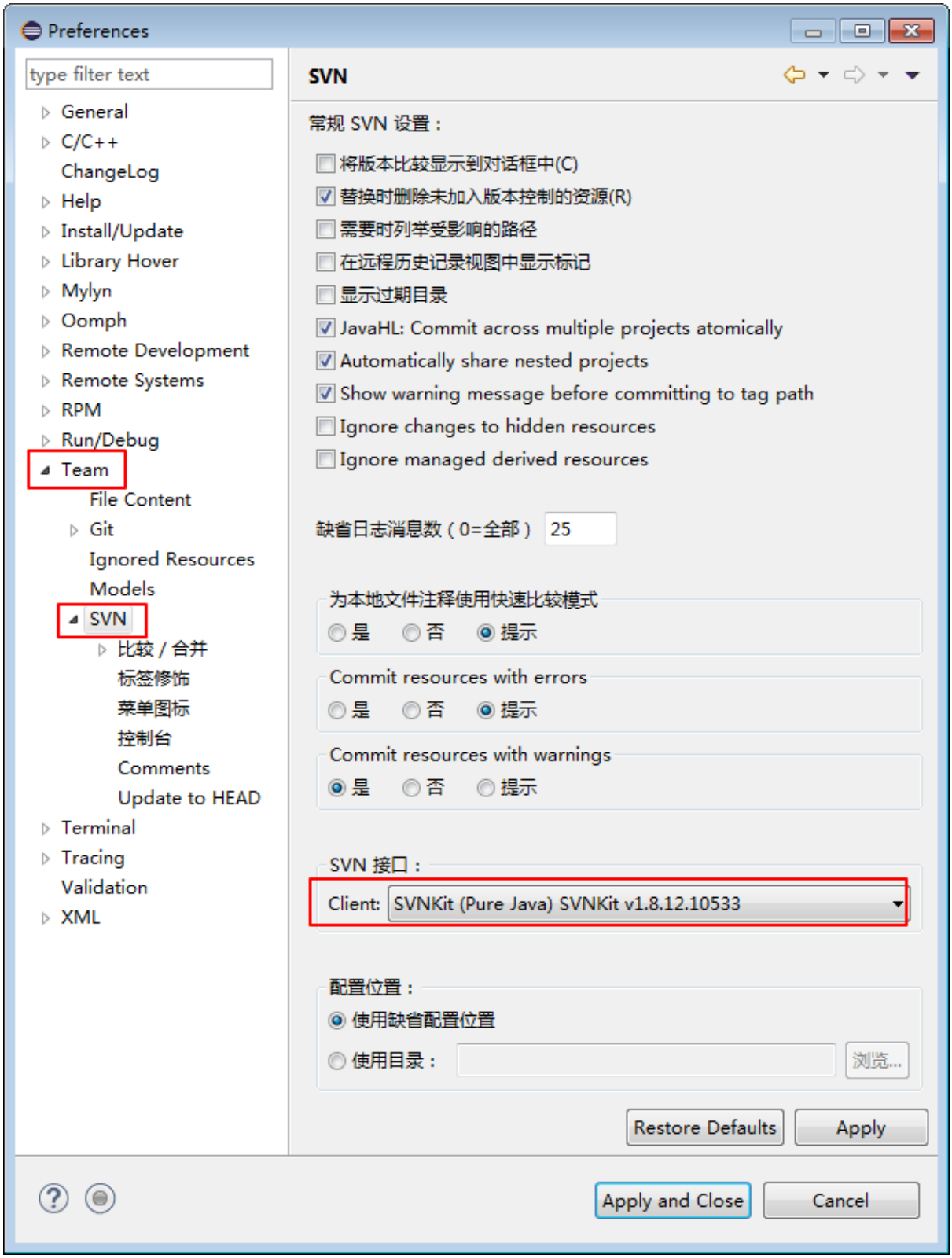
在创建项目之前，我们需要弄清楚两个概念：项目(`Project`)和工作空间(`Workspace`)

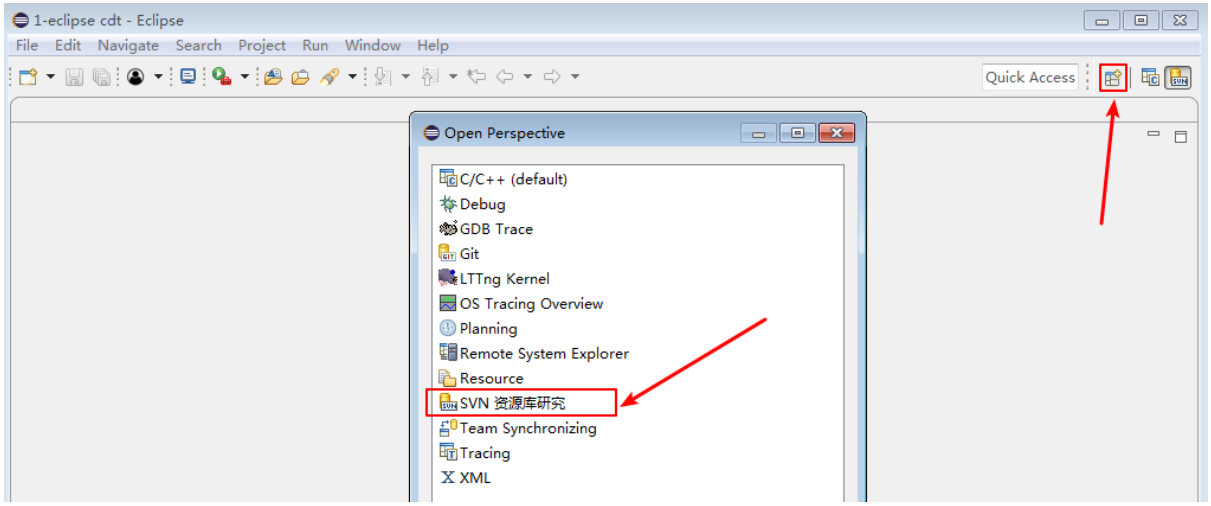
- `Workspace`是`Project`的集合
- `Project`是源代码文件的集合

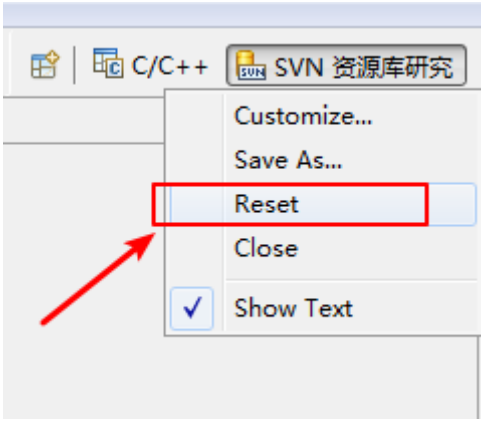
`Eclipse`是通过项目和工作区间的概念来组织源代码的

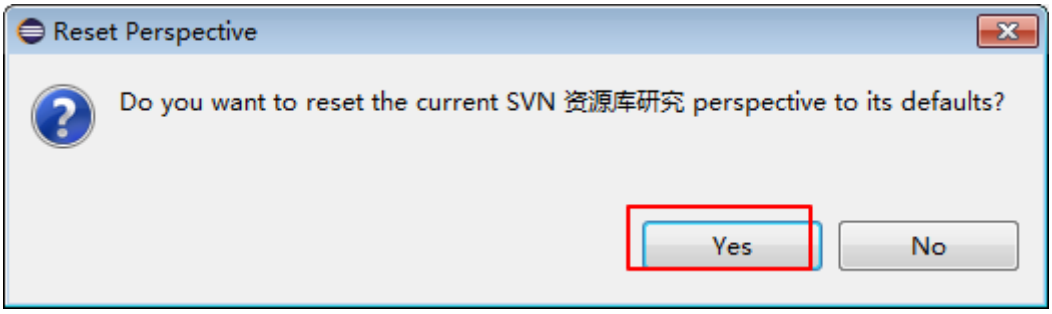
`Workspace`具有以下特点

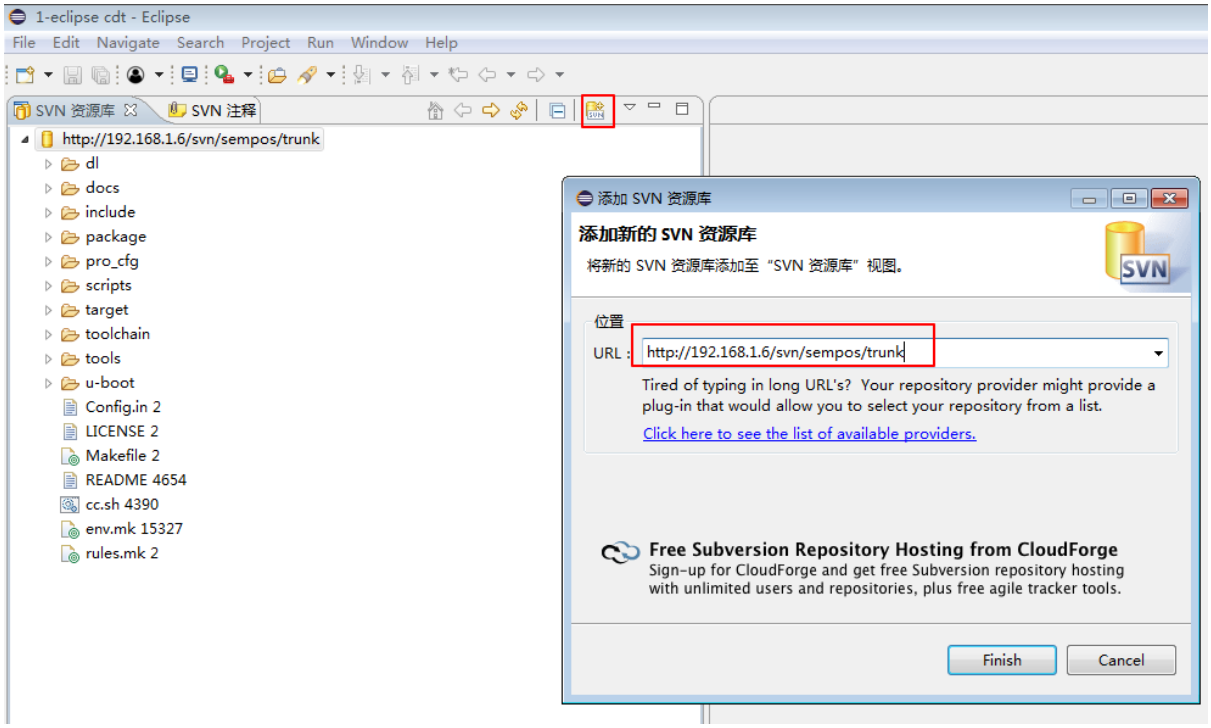
- 该目录下`.metadata`目录存储了该工作空间中所有项目和插件的配置信息(包括语法高亮颜色，字体等)，此目录的存在标识该目录是有效的工作区间
- `.metadata`目录中还包含了以`.log`命名的文件。此文件将包含在运行`Eclipse`时可能抛出的所有错误或异常
- 设置工作空间的方法就是：在首次启动`Eclipse`时，根据提示设置工作空间的默认位置，此时可以不用勾选`Use this as the default and do not ask again`，这样每次启动时都会弹窗显示，根据需要选择；如果首次勾选了该项同时需要修改工作空间，选择`Window->Preferences->General->Startup and Shutdow->Workspace`，勾选`Prompt for workspaces on startup`，最后重启，会和首次启动`Eclipse`一样，弹窗提示设置默认工作区间
- 切换工作空间的方法是：选择`File->Switch Workspace->Other`，可以随时切换到其他工作空间或创建新的工作空间
- 将当前工作空间的所有设置复制到新工作空间中的简单方法
  - 选择`File->Export`


Preferences
type filter text
General
C/C++
ChangeLog
Help
Install/Update
Library Hover
Mylyn
Oomph
Remote Development
Remote Systems
RPM
Run/Debug
Team
File Content
Git
Ignored Resources
Models
SVN
比较 / 合并
标签修饰
菜单图标
控制台
Comments
Update to HEAD
Terminal
Tracing
Validation
XML
SVN
常规 SVN 设置：
将版本比较显示到对话框中(C)
替换时删除未加入版本控制的资源(R)
需要时列举受影响的路径
在远程历史记录视图中显示标记
显示过期目录
JavaHL: Commit across multiple projects atomically
Automatically share nested projects
Show warning message before committing to tag path
Ignore changes to hidden resources
Ignore managed derived resources
缺省日志消息数（0=全部） 25
为本地文件注释使用快速比较模式
是 否 提示
Commit resources with errors
是 否 提示
Commit resources with warnings
是 否 提示
SVN 接口：
Client: SVNKit (Pure Java) SVNKit v1.8.12.10533
配置位置：
使用缺省配置位置
使用目录：
浏览...
Restore Defaults
Apply
Apply and Close
Cancel


1-eclipse cdt - Eclipse
File Edit Navigate Search Project Run Window Help
Quick Access
Open Perspective
C/C++ (default)
Debug
GDB Trace
Git
LTTng Kernel
OS Tracing Overview
Planning
Remote System Explorer
Resource
SVN 资源库研究
Team Synchronizing
Tracing
XML


C/C++
SVN 资源库研究
Customize...
Save As...
Reset
Close
Show Text


Reset Perspective
Do you want to reset the current SVN 资源库研究 perspective to its defaults?
Yes
No



  - 在打开的对话框中，选择General->Preferences，然后单击Next
  - 选择Export All，提供文件的导出路径，然后单击Finish。工作空间的所有设置都将被保存到指定路径中
  - 切换到新工作空间，选择File->Import，然后选择General->Preferences，指向刚刚保存的设置文件并单击Finish。您的设置将被导入到新工作空间中

Project有以下特点

- 项目必须放在工作空间目录下才可以被Eclipse使用
- 项目目录中存放的是源码文件，即独立的应用程序或模块

0x0000 svn仓库检出创建

在eclipse启动时，指定默认工作空间路径(设置工作空间路径参考:*workspace*)

- 工作空间设置为映射区间：适用于检出c项目时，勾选作为工作空间中的项目检出选项，此时项目源码路径就是映射区间路径下，这样可以实现本机和服务器之间的代码同步，然后在服务器上进行编译
- 工作空间设置为映射区间和本地路径都行：适用于检出c项目时，勾选作为新项目检出，并使用新建项目向导进行配置选项，此时项目源码路径可以重新指定为映射区间，也可以实现本机和服务器之间的代码同步，然后在服务器上进行编译

这里我们将默认工作区间设置为映射区间

找到package->semp_system->app目录，右键，选择检出为

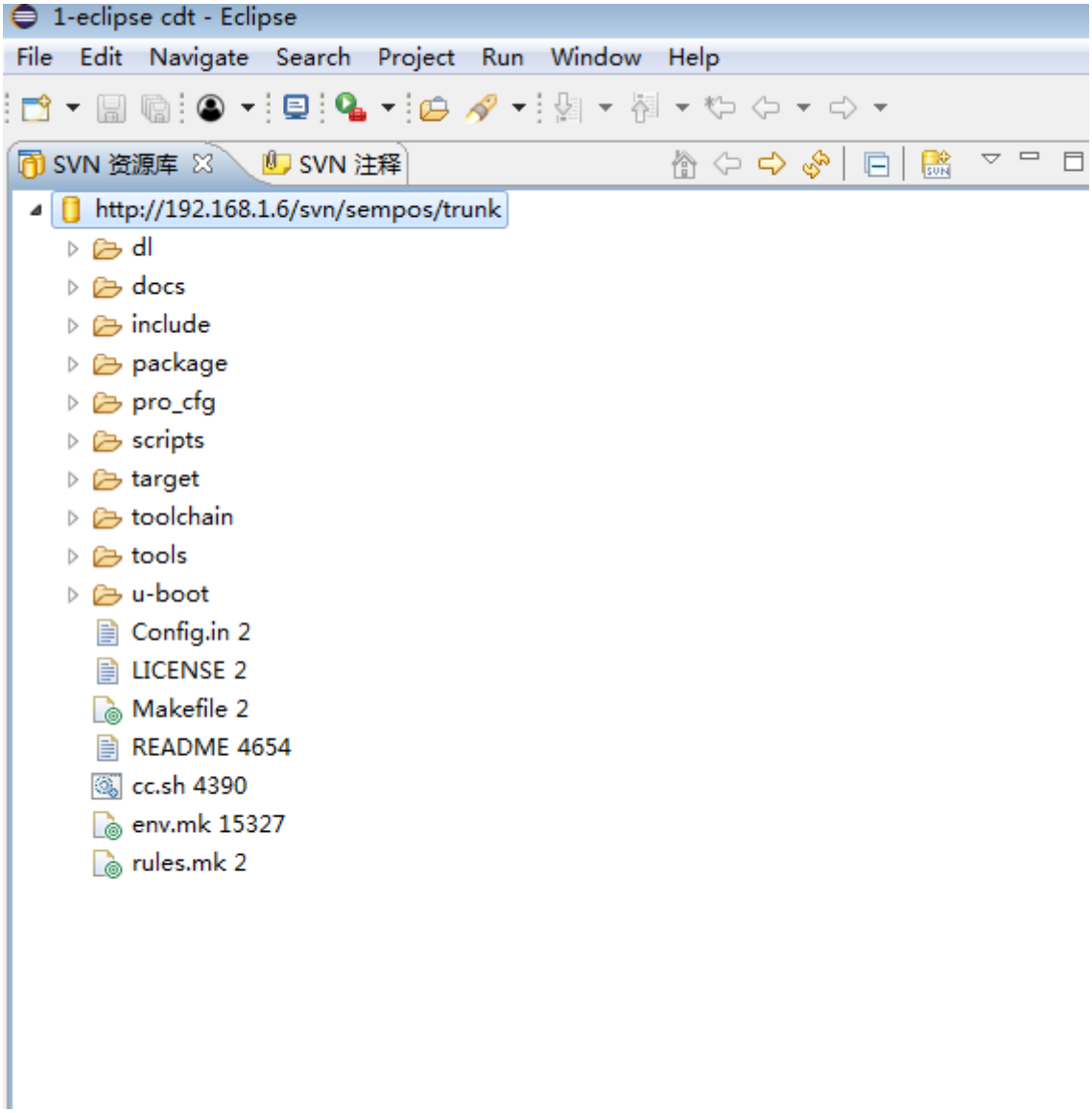
修改项目名称

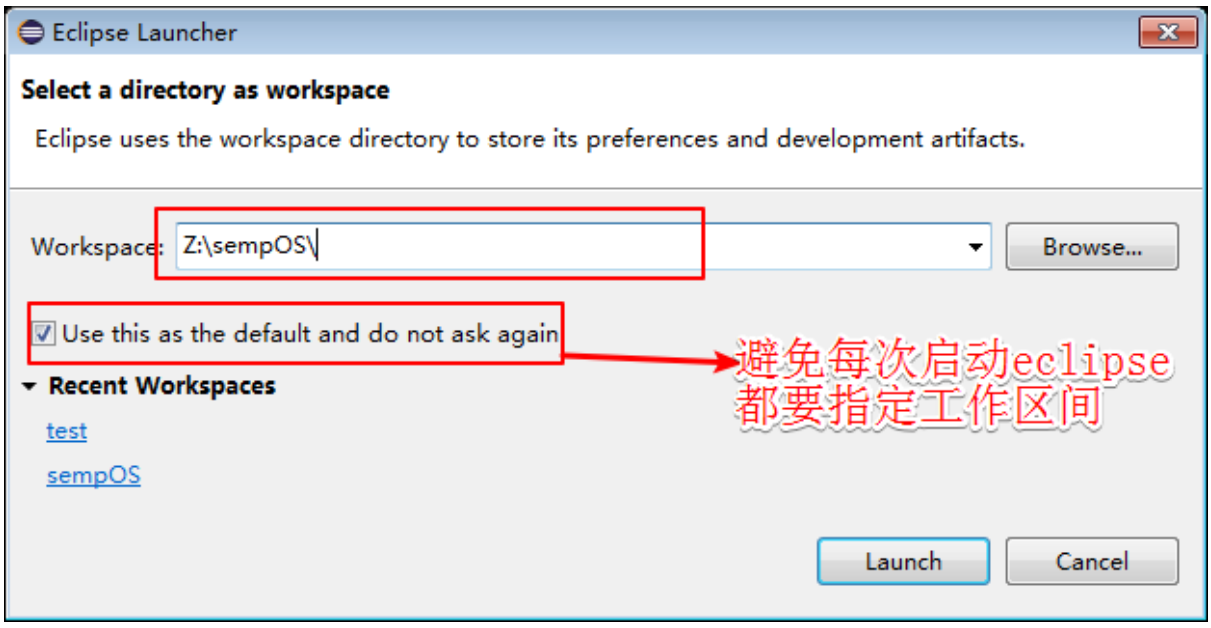
检出完成后，点击右上角Open Perspective按钮，打开c/c++(default)或直接点击右上角的c/c++(default)按钮，就可以看到新建的工程

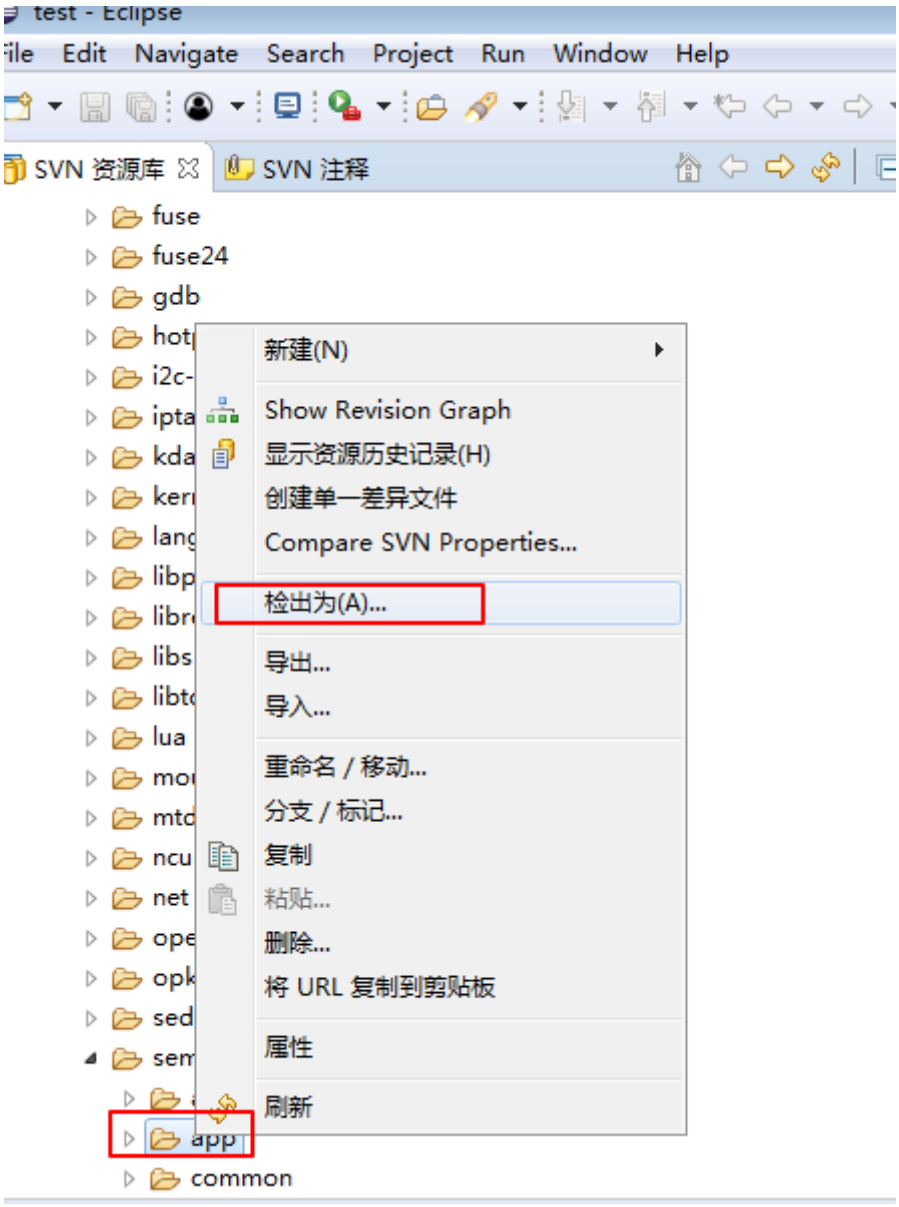
由于检出的文件太多，可能导致eclipse变慢，因此可以删除部分用不到的文件

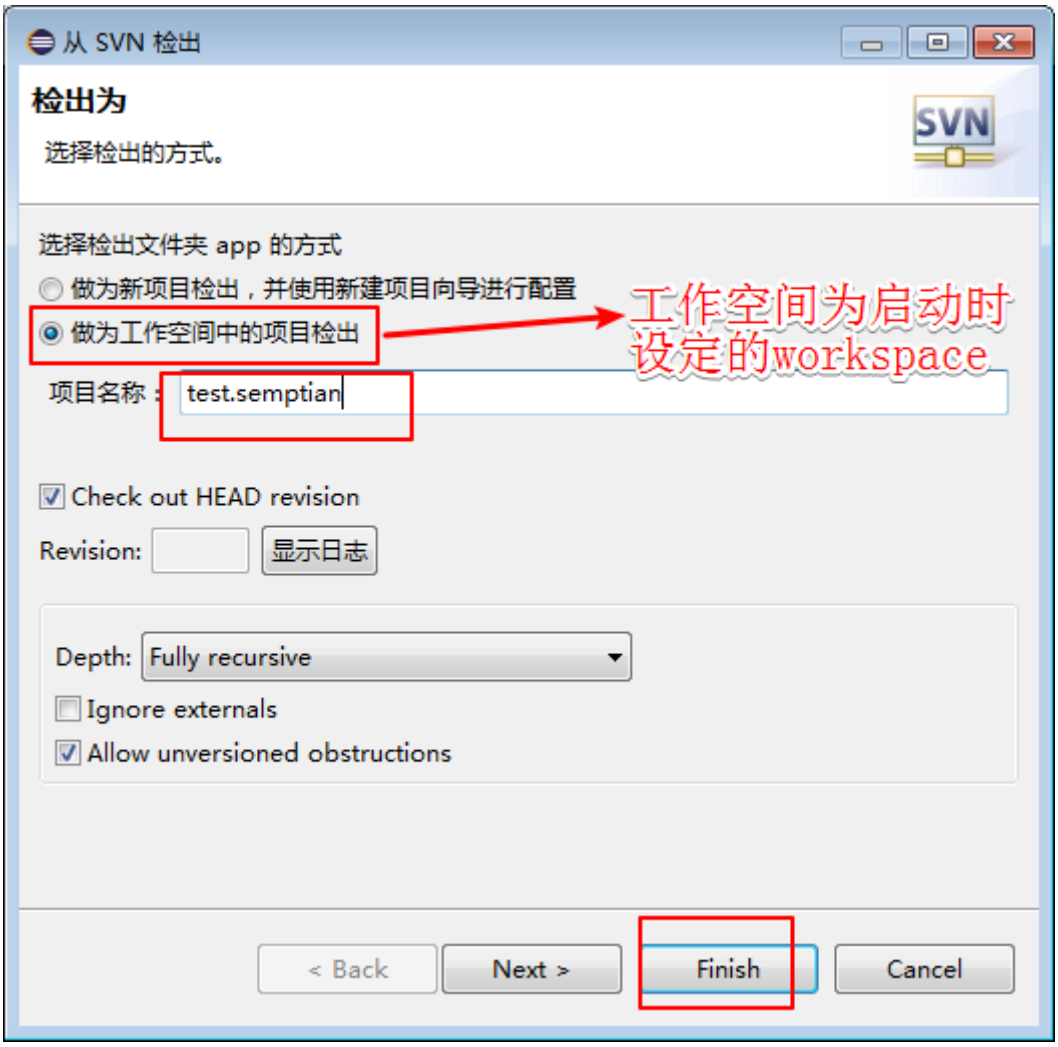
进入工程目录Z:\sempOS\test.sempos，删除下列文件

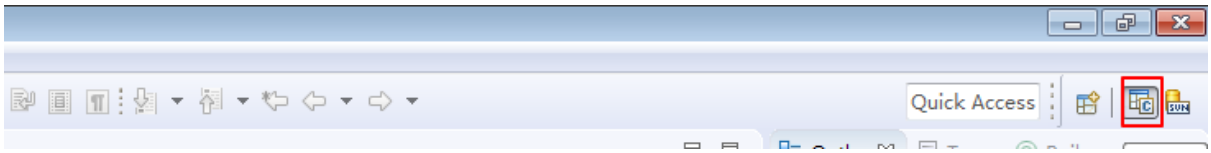
- src/drv/tools

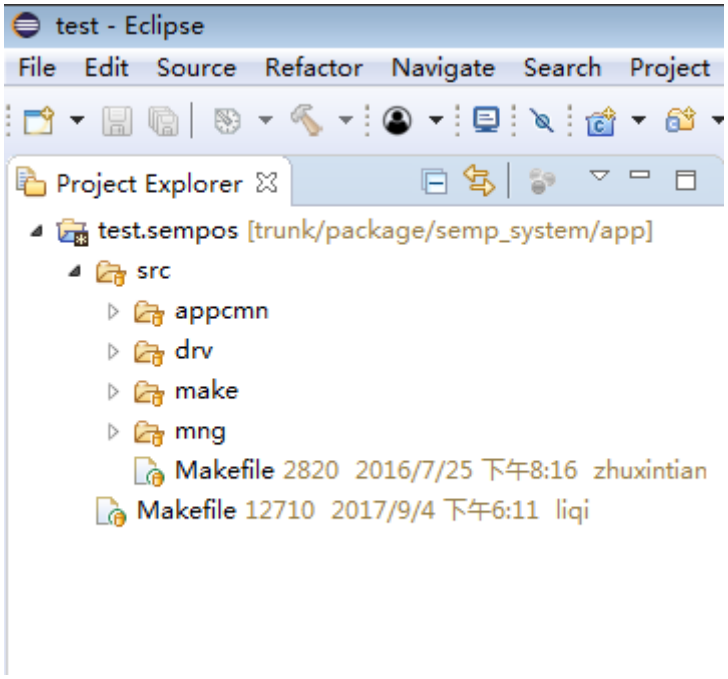

1-eclipse cdt - Eclipse
File Edit Navigate Search Project Run Window Help
SVN 资源库
SVN 注释
http://192.168.1.6/svn/sempos/trunk
dl
docs
include
package
pro_cfg
scripts
target
toolchain
tools
u-boot
Config.in 2
LICENSE 2
Makefile 2
README 4654
cc.sh 4390
env.mk 15327
rules.mk 2


Eclipse Launcher
Select a directory as workspace
Eclipse uses the workspace directory to store its preferences and development artifacts.
Workspace:
Z:\sempOS\
Browse...
Use this as the default and do not ask again
避免每次启动eclipse
都要指定工作区间
Recent Workspaces
test
sempOS
Launch
Cancel


test - Eclipse
File Edit Navigate Search Project Run Window Help
SVN 资源库
SVN 注释
fuse
fuse24
gdb
新建(N)
Show Revision Graph
显示资源历史记录(H)
创建单一差异文件
Compare SVN Properties...
检出为(A)...
导出...
导入...
重命名 / 移动...
分支 / 标记...
复制
粘贴...
删除...
将 URL 复制到剪贴板
属性
刷新
app
common

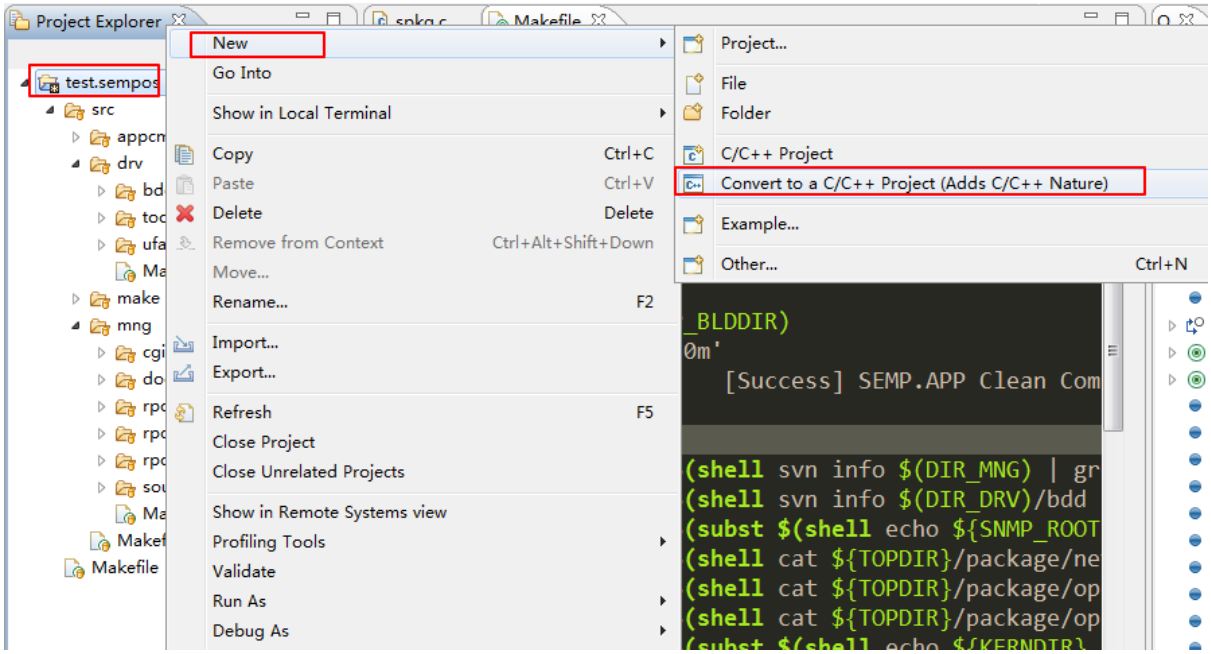

从 SVN 检出
检出为
选择检出的方式。
SVN
选择检出文件夹 app 的方式
做为新项目检出，并使用新建项目向导进行配置
做为工作空间中的项目检出
工作空间为启动时
设定的workspace
项目名称：
test.semptian
Check out HEAD revision
Revision:
显示日志
Depth:
Fully recursive
Ignore externals
Allow unversioned obstructions
< Back
Next >
Finish
Cancel


Quick Access



- `src/drv/ufal`
- `src/drv/bdd/dc6408`
- `src/mng/doc`

右键选中项目，选择`New->Convert to a C/C++ Project`

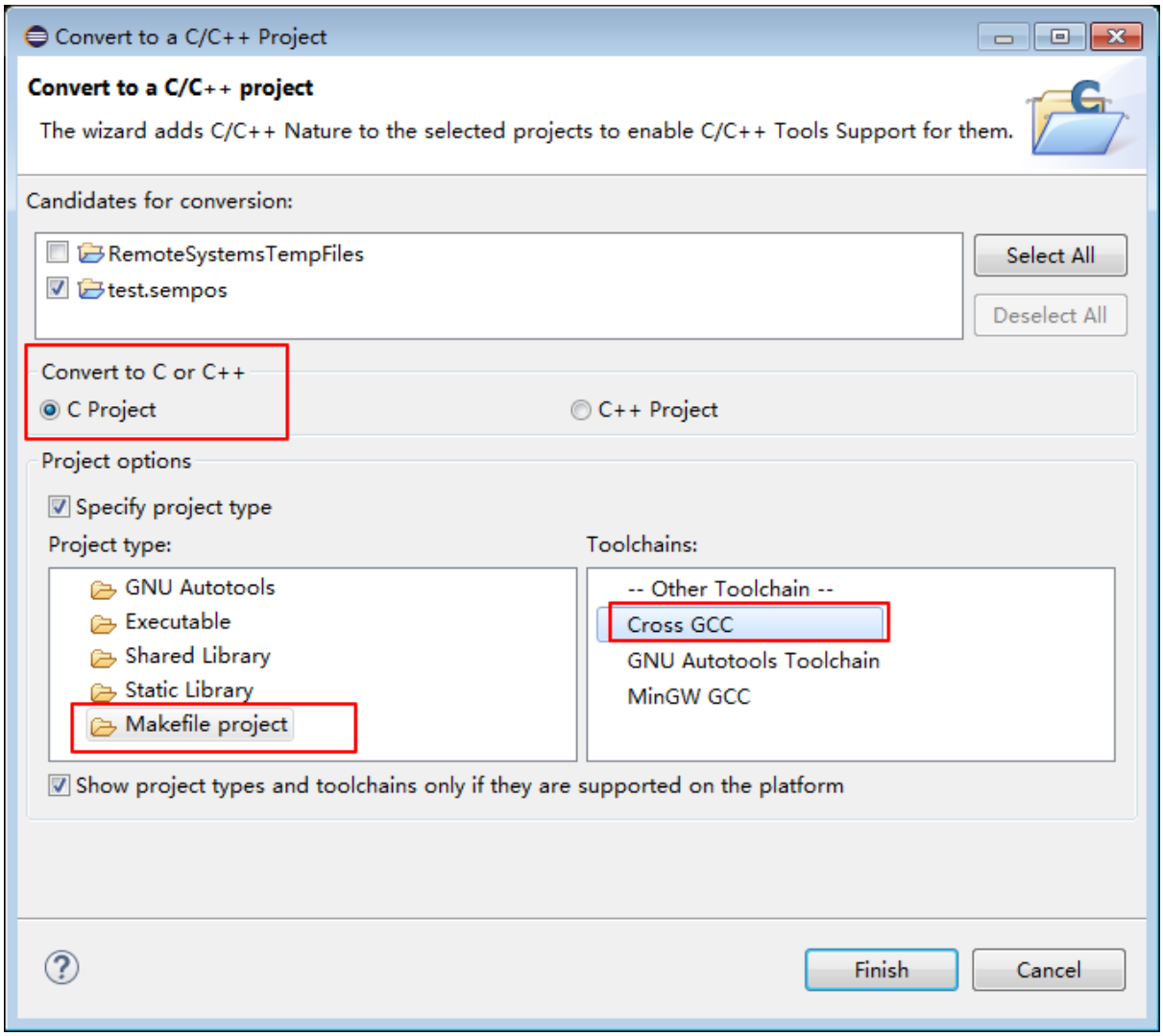


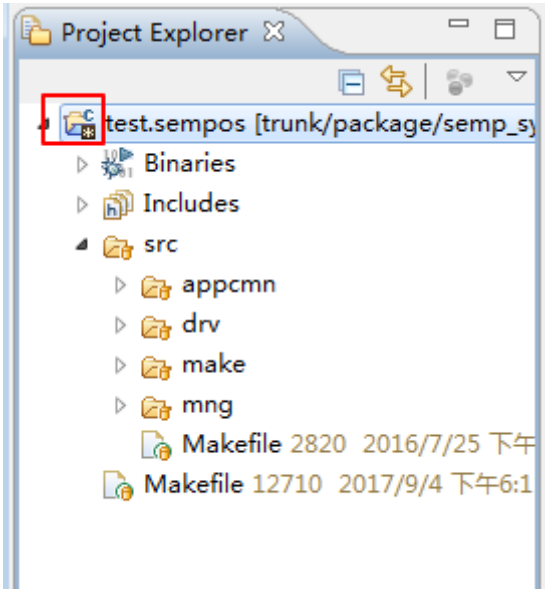
选择转换为C项目，项目类型为`Makefile`，工具链选`Cross GCC`

转换完成后，项目图标会变为C

0x0001 eclipse手动新建

在服务器上创建一个目录，用来单独存放所有项目的源码；然后使用svn命令下载源码到该目录下，重


Convert to a C/C++ Project
Convert to a C/C++ project
The wizard adds C/C++ Nature to the selected projects to enable C/C++ Tools Support for them.
Candidates for conversion:
RemoteSystemsTempFiles
test.sempos
Select All
Deselect All
Convert to C or C++
C Project
C++ Project
Project options
Specify project type
Project type:
GNU Autotools
Executable
Shared Library
Static Library
Makefile project
Toolchains:
-- Other Toolchain --
Cross GCC
GNU Autotools Toolchain
MinGW GCC
Show project types and toolchains only if they are supported on the platform
Finish
Cancel

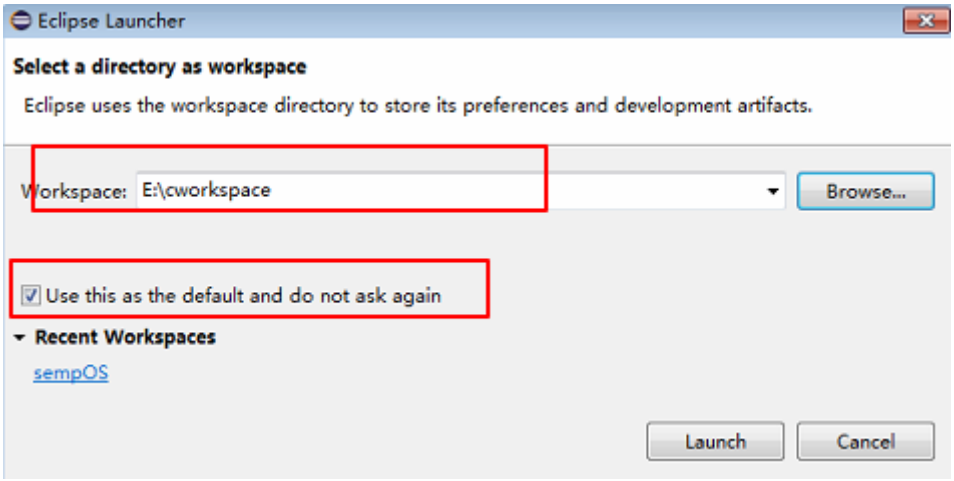

Project Explorer
test.sempos [trunk/package/semp_sy
Binaries
Includes
src
appcmn
drv
make
mng
Makefile 2820 2016/7/25 下午
Makefile 12710 2017/9/4 下午6:1

命名项目名称

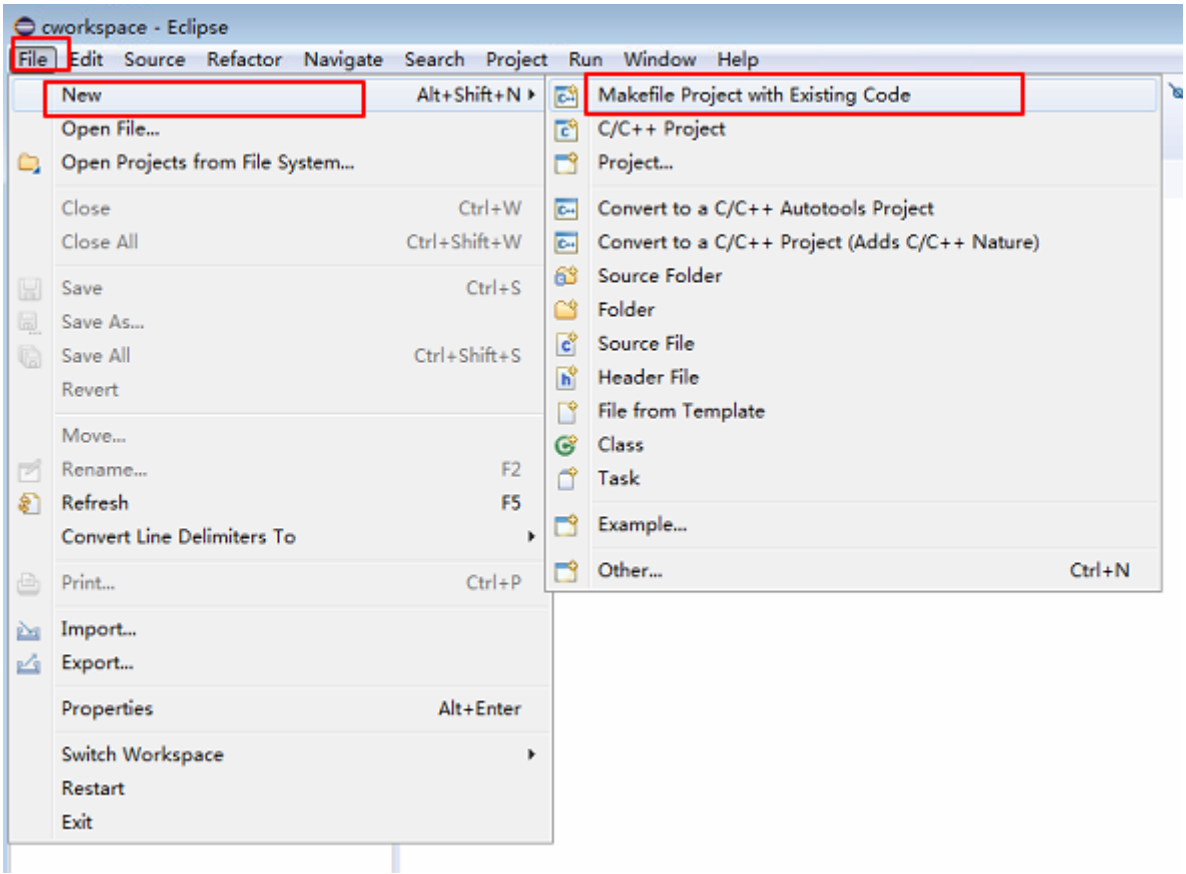
- `mkdir sempOS && cd sempOS`
- `svn co http://192.168.1.6/svn/sempos/trunk`
- `mv -r trunk/ DC6408_RPC`

在eclipse启动时，指定默认工作空间路径为映射区间和本地路径都行；因为手动创建c项目时，项目源码路径可以重新指定为映射区间，实现本机和服务器之间的代码同步，然后在服务器上进行编译

这里我们将默认工作区间设置为本地路径(设置工作空间路径参考:*workspace*)

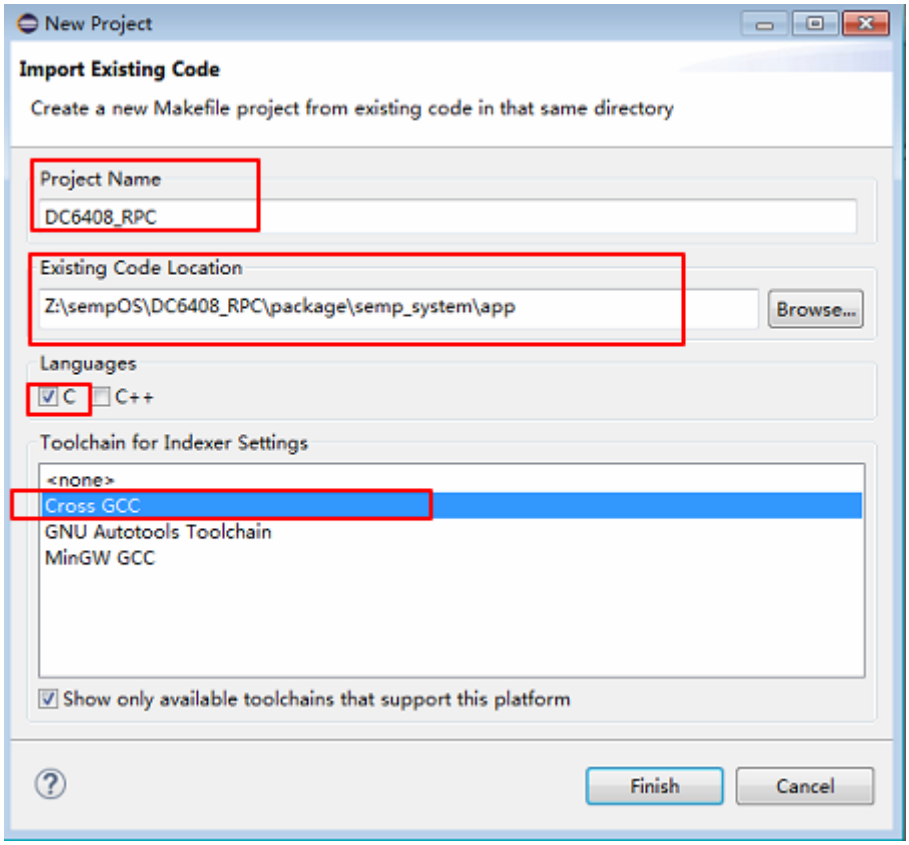


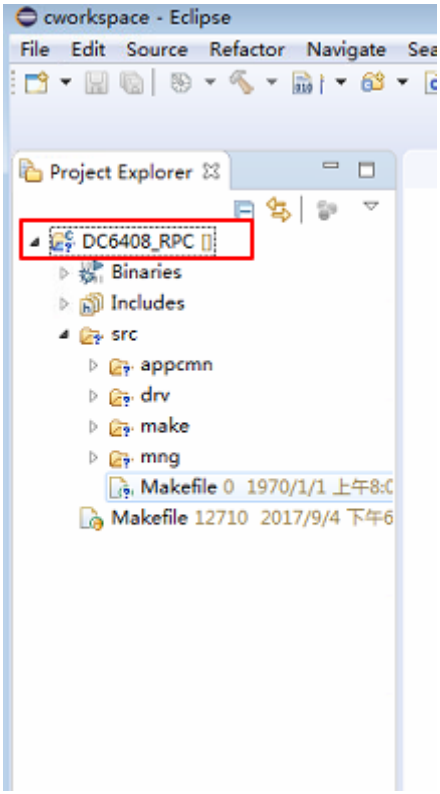
选择`File->New->Makefile Project with Existing Code`



指定项目名称，已存在项目源码路径，选择C语言和`Cross Gcc`工具链

创建项目成功后如下所示，项目图标会变为C


New Project
Import Existing Code
Create a new Makefile project from existing code in that same directory
Project Name
DC6408_RPC
Existing Code Location
Z:\sempOS\DC6408_RPC\package\semp_system\app
Browse...
Languages
C
C++
Toolchain for Indexer Settings
<none>
Cross GCC
GNU Autotools Toolchain
MinGW GCC
Show only available toolchains that support this platform
Finish
Cancel


cworkspace - Eclipse
File
Edit
Source
Refactor
Navigate
Project Explorer
DC6408_RPC
Binaries
Includes
src
appcmn
drv
make
mng
Makefile 0 1970/1/1 上午8:0
Makefile 12710 2017/9/4 下午6

0x01 设置项目选项

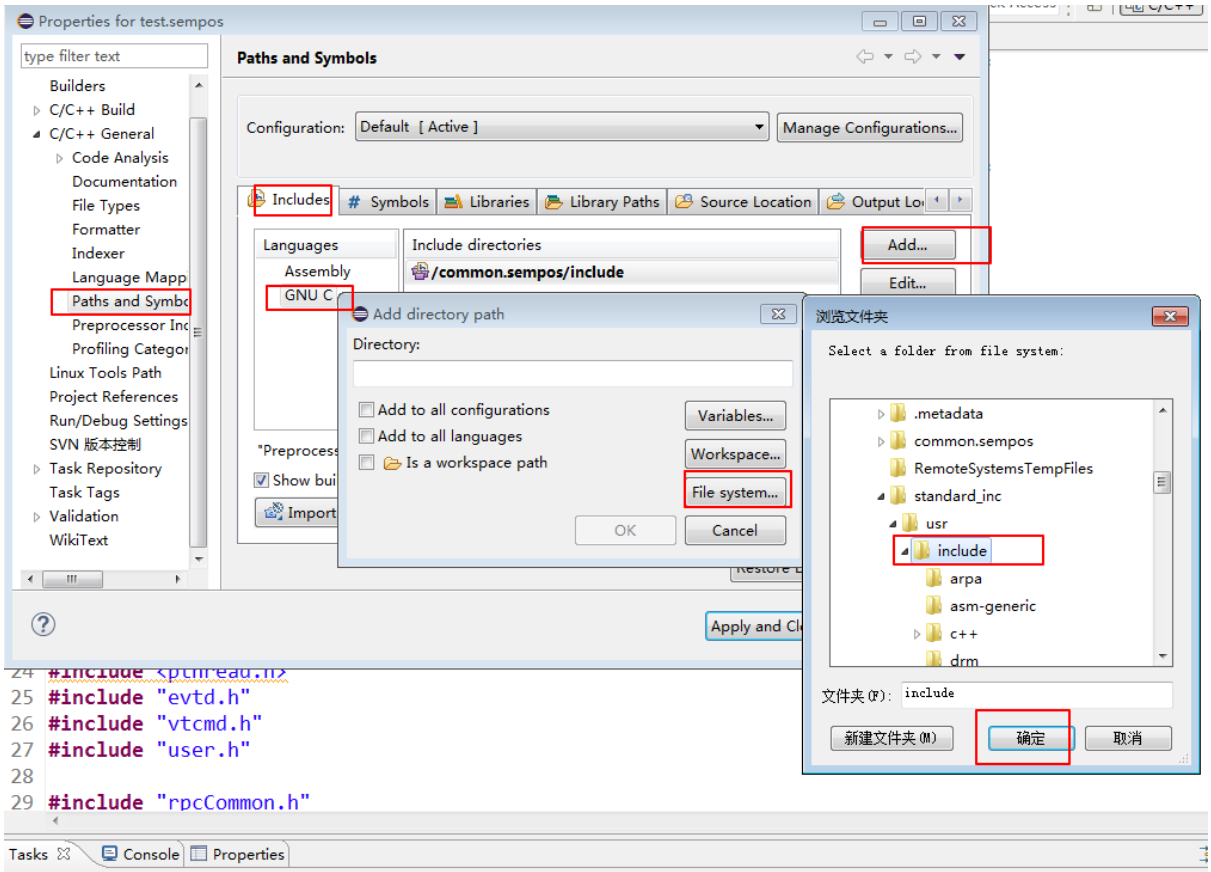
如果在项目中需要实现函数、变量等的跳转，项目的include头文件路径中需要包含以下路径下的头文件

- /usr/include
- /usr/include/x86_64-linux-gnu
- /usr/local/include
- /usr/lib/gcc/x86_64-linux-gnu/4.8/include
- trunk/package/semp_system/common/include

对于前4类头文件，需要将这些目录下的头文件拷贝到映射区间，然后添加到eclipse中

- mkdir -p ~/sempOS/standard_inc/usr/{local,lib/gcc/x86_64-linux-gnu/4.8}
- cp -r /usr/include ~/sempOS/standard_inc/usr
- cp -r /usr/local/include ~/sempOS/standard_inc/usr/local
- cp -r /usr/lib/gcc/x86_64-linux-gnu/4.8/include ~/sempOS/standard_inc/usr/lib/gcc/x86_64-linux-gnu/4.8

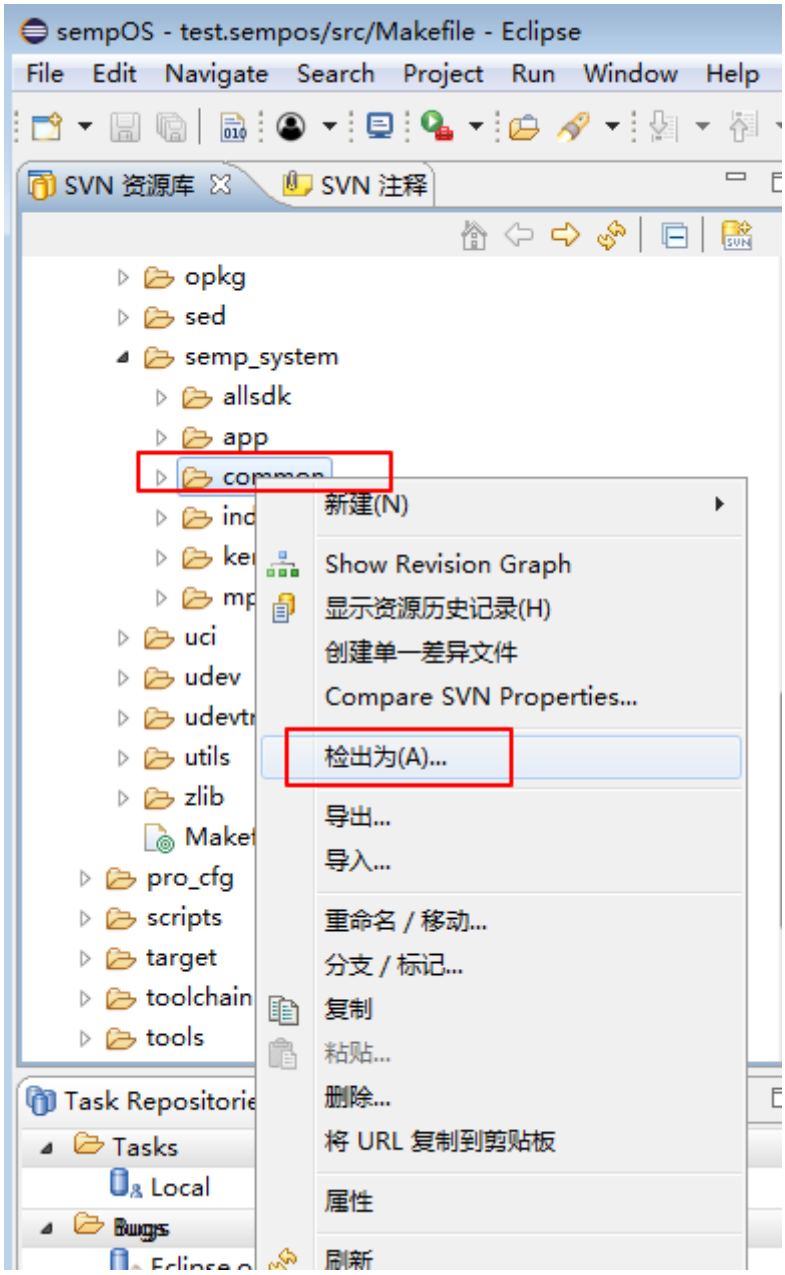
右键选中项目，选择Properties，修改C/C++ General下的Paths and Symbols设置，将上述头文件的路径添加到include头文件包含路径中

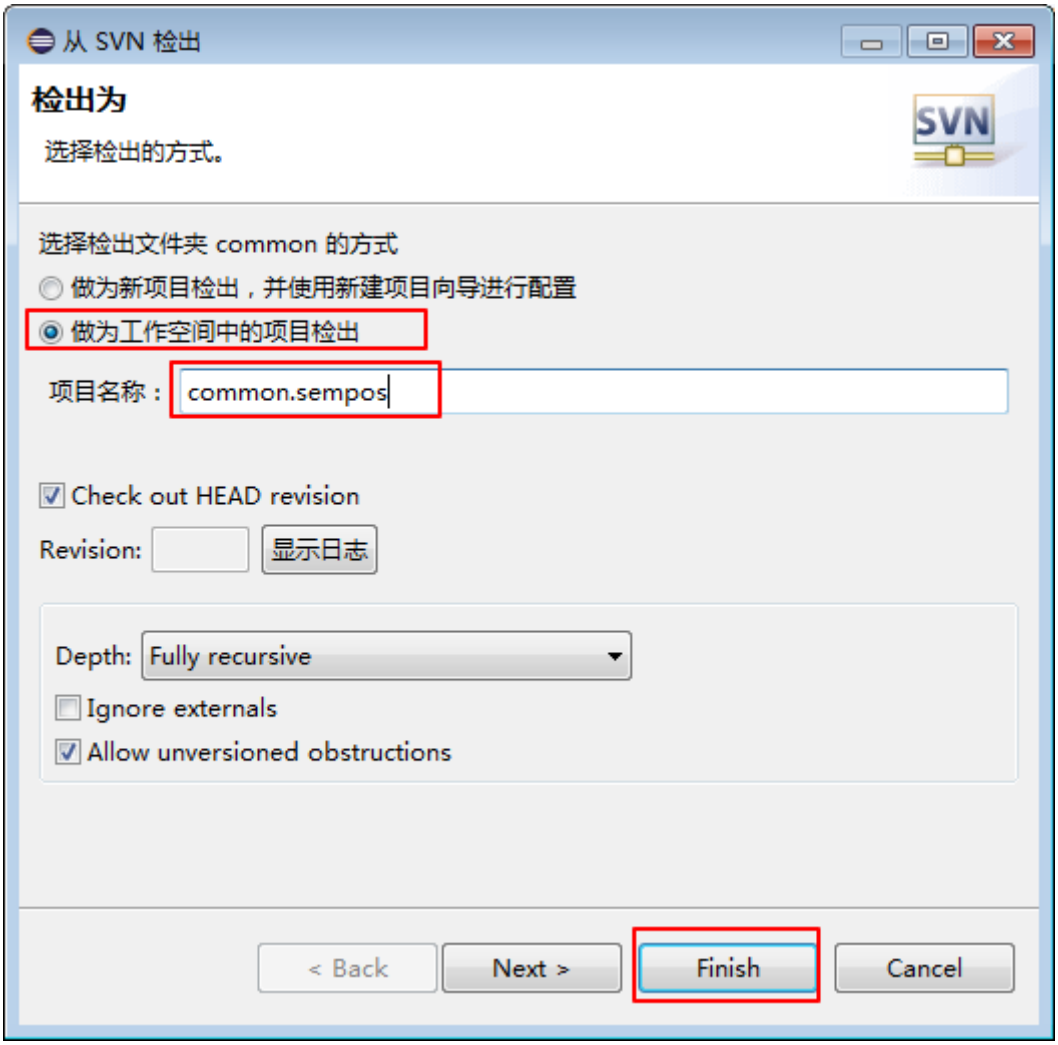


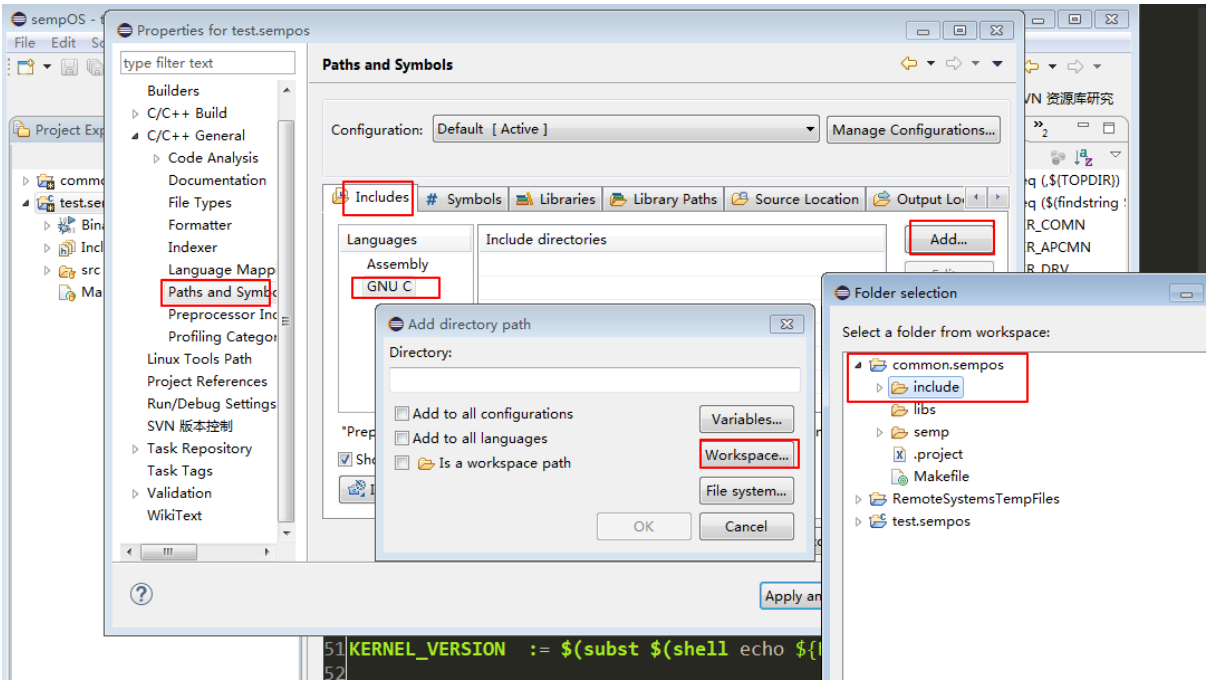
对于最后一类头文件

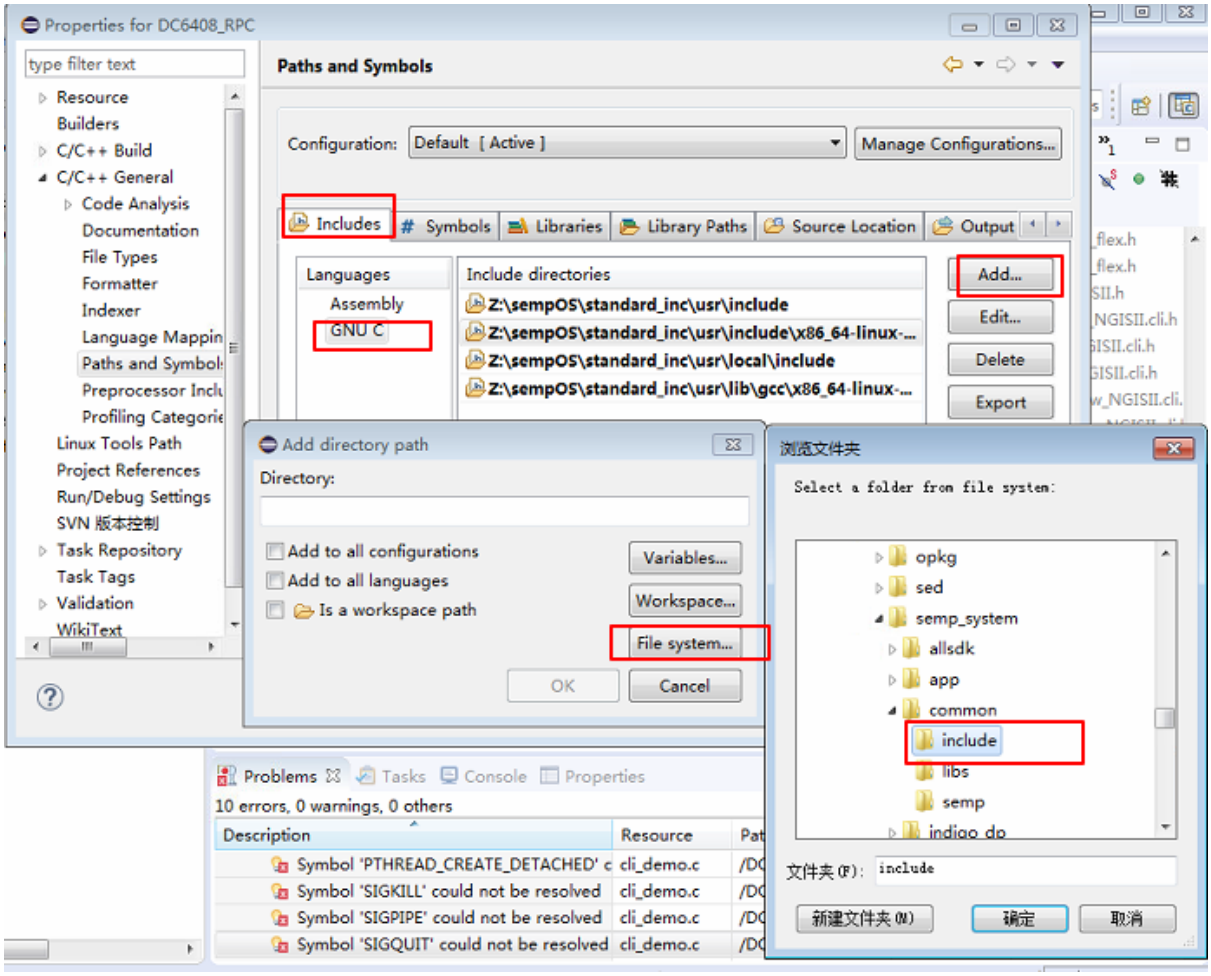
- 使用svn仓库检出方法创建项目时，需要将common目录检出为一个新的工程然后添加
- 使用eclipse手动新建方法创建项目时，只需要添加下载源码common目录下的头文件即可

添加完成后如下：

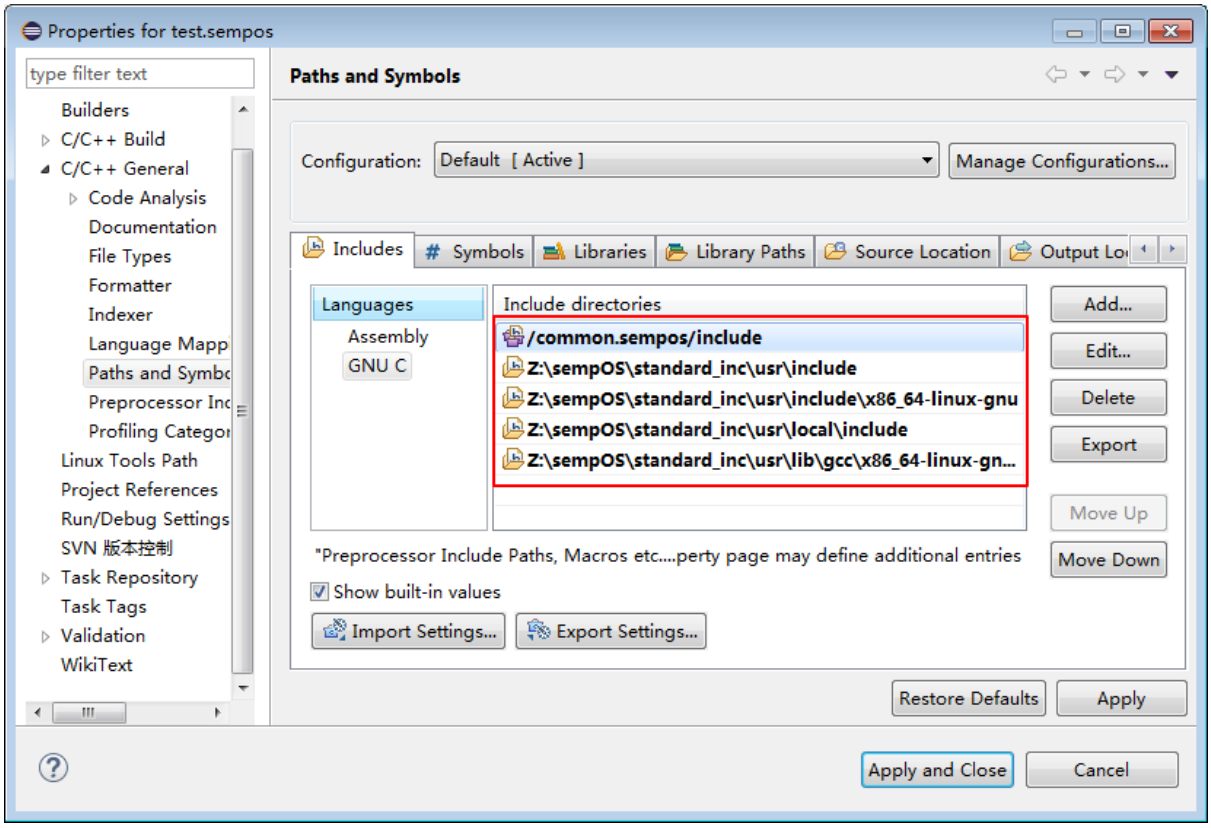

sempOS - test.sempos/src/Makefile - Eclipse
File Edit Navigate Search Project Run Window Help
SVN 资源库
SVN 注释
opkg
sed
semp_system
allsdk
app
uci
udev
utils
zlib
pro_cfg
scripts
target
toolchain
tools
新建(N)
Show Revision Graph
显示资源历史记录(H)
创建单一差异文件
Compare SVN Properties...
检出为(A)...
导出...
导入...
重命名 / 移动...
分支 / 标记...
复制
粘贴...
删除...
将 URL 复制到剪贴板
属性
刷新
Task Repositorie
Tasks
Local


从 SVN 检出
检出为
选择检出的方式。
SVN
选择检出文件夹 common 的方式
做为新项目检出，并使用新建项目向导进行配置
做为工作空间中的项目检出
项目名称：
common.sempos
Check out HEAD revision
Revision:
显示日志
Depth:
Fully recursive
Ignore externals
Allow unversioned obstructions
< Back
Next >
Finish
Cancel


Properties for test.sempos
type filter text
Builders
C/C++ Build
C/C++ General
Code Analysis
Documentation
File Types
Formatter
Indexer
Language Mapp
Paths and Symb
Preprocessor Inc
Profiling Categor
Linux Tools Path
Project References
Run/Debug Settings
SVN 版本控制
Task Repository
Task Tags
Validation
WikiText
Paths and Symbols
Configuration:
Default [Active]
Manage Configurations...
Includes
Symbols
Libraries
Library Paths
Source Location
Output Lo
Languages
Assembly
GNU C
Include directories
Add...
Add directory path
Directory:
Add to all configurations
Add to all languages
Is a workspace path
Variables...
Workspace...
File system...
OK
Cancel
Folder selection
Select a folder from workspace:
common.sempos
include
libs
semp
.project
Makefile
RemoteSystemsTempFiles
test.sempos
KERNEL_VERSION := $(subst $(shell echo


Properties for DC6408_RPC
type filter text
Resource
Builders
C/C++ Build
C/C++ General
Code Analysis
Documentation
File Types
Formatter
Indexer
Language Mappin
Paths and Symbol
Preprocessor Incl
Profiling Categorie
Linux Tools Path
Project References
Run/Debug Settings
SVN 版本控制
Task Repository
Task Tags
Validation
WikiText
Paths and Symbols
Configuration: Default [Active]
Manage Configurations...
Includes
Symbols
Libraries
Library Paths
Source Location
Output
Languages
Assembly
GNU C
Include directories
Z:\sempOS\standard_inc\usr\include
Z:\sempOS\standard_inc\usr\include\x86_64-linux-...
Z:\sempOS\standard_inc\usr\local\include
Z:\sempOS\standard_inc\usr\lib\gcc\x86_64-linux-...
Add...
Edit...
Delete
Export
Add directory path
Directory:
Add to all configurations
Add to all languages
Is a workspace path
Variables...
Workspace...
File system...
OK
Cancel
浏览文件夹
Select a folder from file system:
opkg
sed
semp_system
allsdk
app
common
include
libs
semp
indigo_dp
文件夹(F):
include
新建文件夹(N)
确定
取消
Problems
Tasks
Console
Properties
10 errors, 0 warnings, 0 others
Description
Resource
Symbol 'PTHREAD_CREATE_DETACHED' c
Symbol 'SIGKILL' could not be resolved
Symbol 'SIGPIPE' could not be resolved
Symbol 'SIGQUIT' could not be resolved
cli_demo.c

- 使用svn仓库检出方法创建项目效果图



- 使用eclipse手动新建方法创建项目效果图

注意：在有些源码文件还需要包含其它的头文件才可以，例如：`cli_demo.c`源文件，需要包含`/usr/include/x86_64-linux-gnu/asm/`头文件路径，因为该目录下的`signal.h`头文件声明了`SIGKILL`、`SIGPIPE`等宏，如果不包含就会报错显示。此时需要灵活处理，可以使用以下命令查看报错字段声明的头文件，然后包含该头文件路径即可

- `grep -rn "SIGKILL" ~/sempOS/standard_inc/usr`

根据需求，可以在`Eclipse`中添加符号定义，效果相当于Makefile中定义的符号；右键选中项目，选择`Properties`，修改`C/C++ General`下的`Paths and Symbols`设置；点击`Symbols`项，添加自定义符号

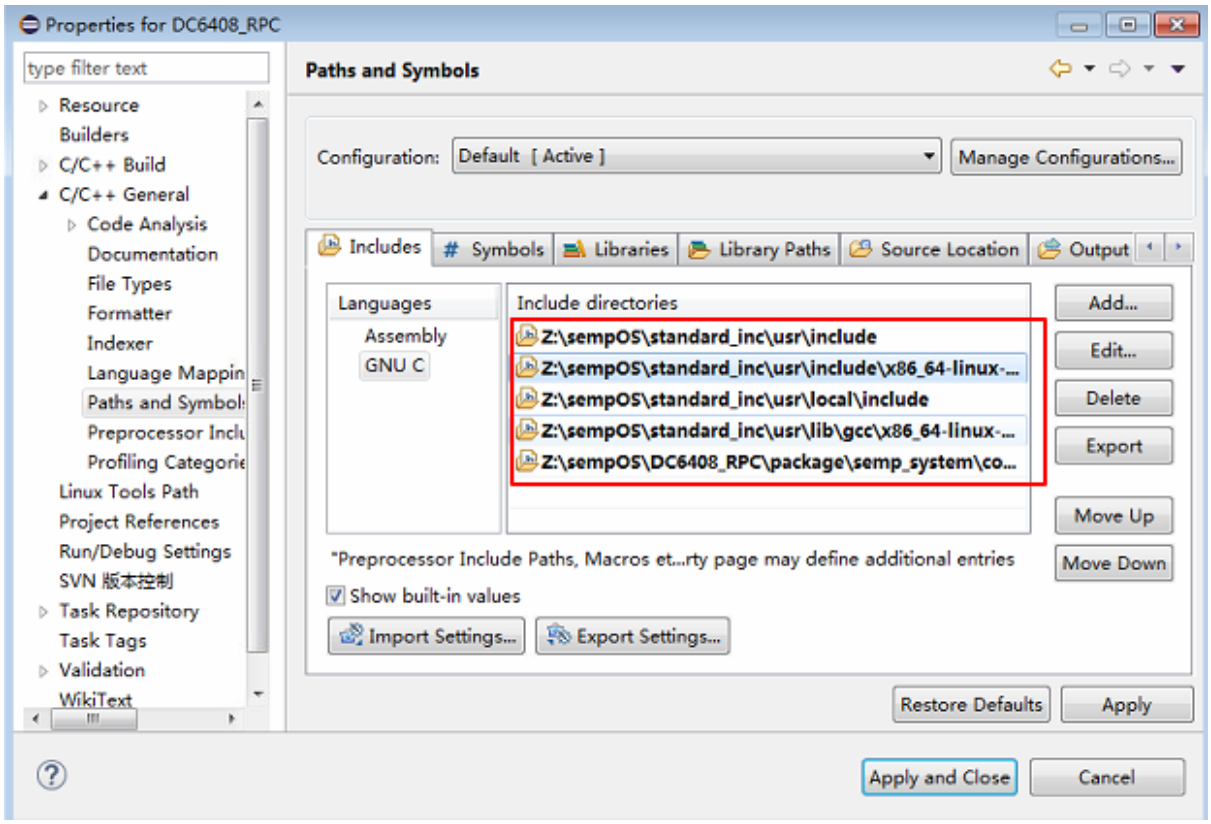
为避免不需要分析的代码，加快速度，可以添加代码路径过滤条件；在`Paths and Symbols`设置；点击`Source Location`项

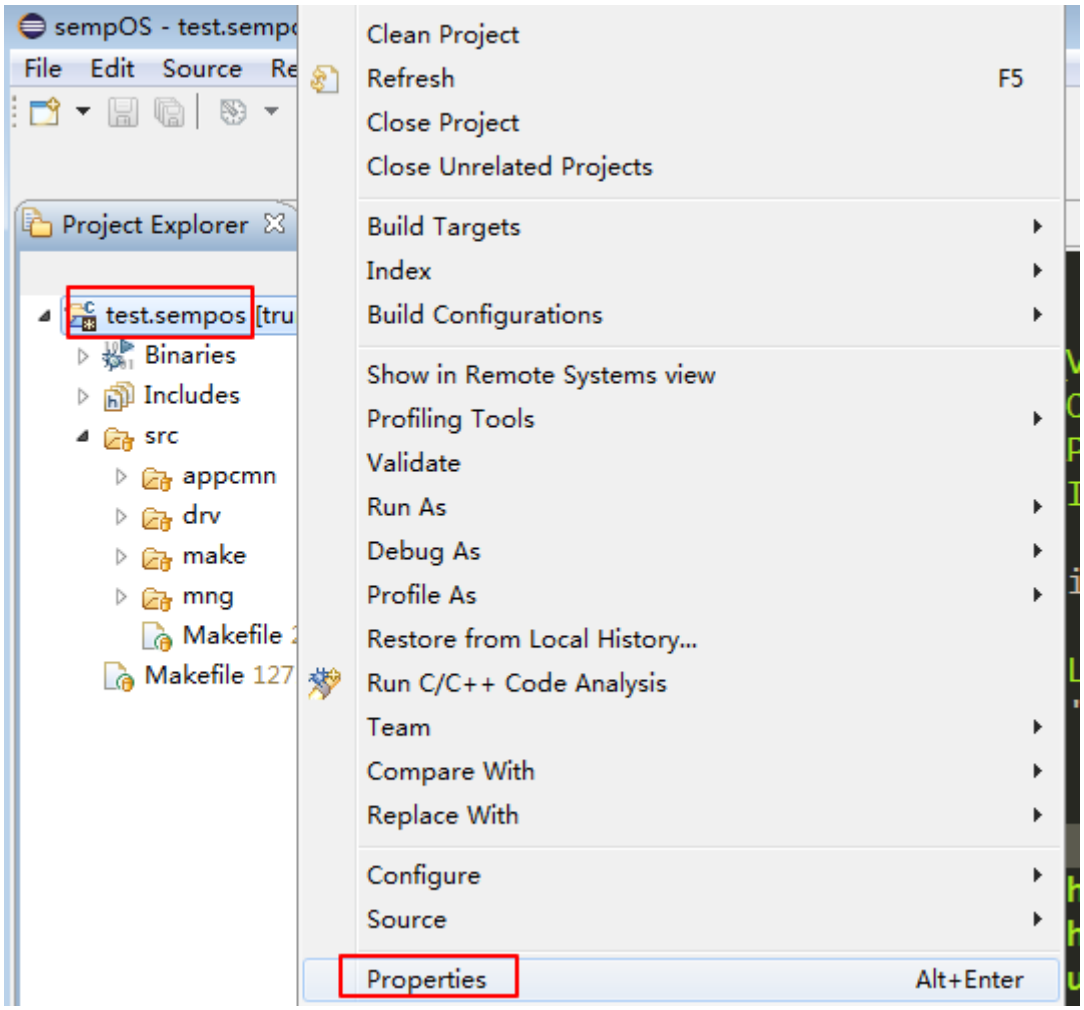
最后右键选中项目，选择`Index->Rebuild`重新建立索引，至此工程选项全部设置完成

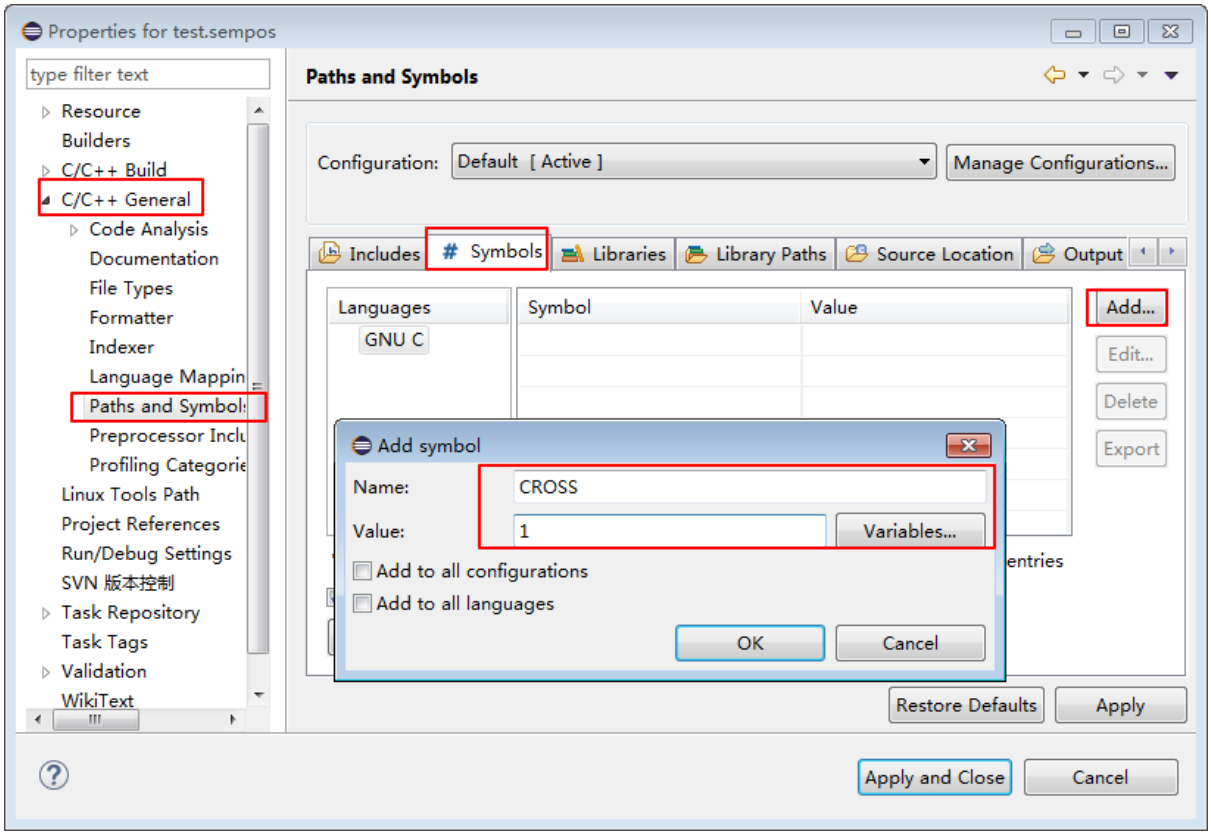
0x02 代码编辑

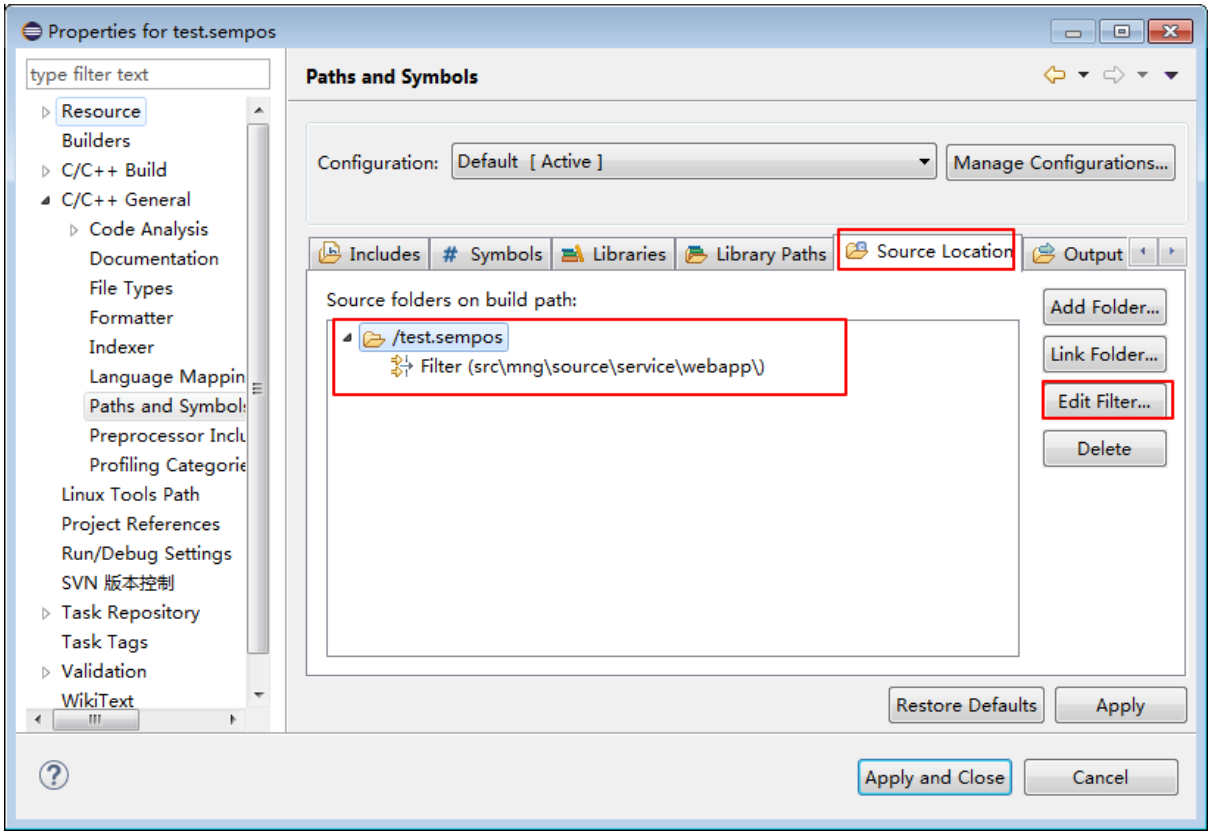
默认快捷键

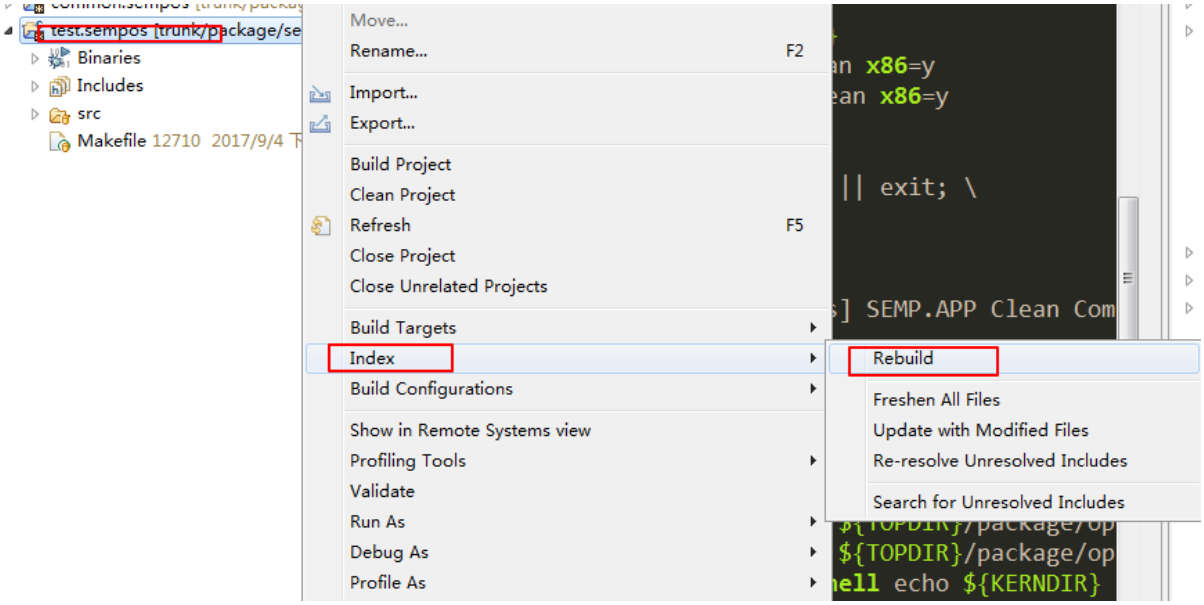
- 回退：`alt+left`
- 前进：`alt+right`
- 代码注释：`ctrl+/`
- 自动补全：`alt+/`
- 代码格式化：`ctrl+shift+F`
- 跳转到调用的地方
  - `ctrl+shift+G`


Properties for DC6408_RPC
type filter text
Resource
Builders
C/C++ Build
C/C++ General
Code Analysis
Documentation
File Types
Formatter
Indexer
Language Mappin
Paths and Symbol
Preprocessor Inclu
Profiling Categorie
Linux Tools Path
Project References
Run/Debug Settings
SVN 版本控制
Task Repository
Task Tags
Validation
WikiText
Paths and Symbols
Configuration: Default [Active]
Manage Configurations...
Includes
Symbols
Libraries
Library Paths
Source Location
Output
Languages
Assembly
GNU C
Include directories
Z:\sempOS\standard_inc\usr\include
Z:\sempOS\standard_inc\usr\include\x86_64-linux-...
Z:\sempOS\standard_inc\usr\local\include
Z:\sempOS\standard_inc\usr\lib\gcc\x86_64-linux-...
Z:\sempOS\DC6408_RPC\package\semp_system\co...
Add...
Edit...
Delete
Export
Move Up
Move Down
*Preprocessor Include Paths, Macros et...rty page may define additional entries
Show built-in values
Import Settings...
Export Settings...
Restore Defaults
Apply
Apply and Close
Cancel


sempOS - test.sempo
File Edit Source Re
Project Explorer
test.sempos [tru
Binaries
Includes
src
appcmn
drv
make
mng
Makefile
Makefile 127
Clean Project
Refresh F5
Close Project
Close Unrelated Projects
Build Targets
Index
Build Configurations
Show in Remote Systems view
Profiling Tools
Validate
Run As
Debug As
Profile As
Restore from Local History...
Run C/C++ Code Analysis
Team
Compare With
Replace With
Configure
Source
Properties Alt+Enter


Properties for test.sempos
type filter text
Resource
Builders
C/C++ Build
C/C++ General
Code Analysis
Documentation
File Types
Formatter
Indexer
Language Mappin
Paths and Symbol
Preprocessor Inclu
Profiling Categorie
Linux Tools Path
Project References
Run/Debug Settings
SVN 版本控制
Task Repository
Task Tags
Validation
WikiText
Paths and Symbols
Configuration: Default [Active]
Manage Configurations...
Includes
Symbols
Libraries
Library Paths
Source Location
Output
Languages
Symbol
Value
GNU C
Add...
Edit...
Delete
Export
Add symbol
Name: CROSS
Value: 1
Variables...
Add to all configurations
Add to all languages
OK
Cancel
Restore Defaults
Apply
Apply and Close
Cancel


Properties for test.sempos
type filter text
Resource
Builders
C/C++ Build
C/C++ General
Code Analysis
Documentation
File Types
Formatter
Indexer
Language Mappin
Paths and Symbol
Preprocessor Inclu
Profiling Categorie
Linux Tools Path
Project References
Run/Debug Settings
SVN 版本控制
Task Repository
Task Tags
Validation
WikiText
Paths and Symbols
Configuration: Default [Active]
Manage Configurations...
Includes
Symbols
Libraries
Library Paths
Source Location
Output
Source folders on build path:
/test.sempos
Filter (src\mng\source\service\webapp\)
Add Folder...
Link Folder...
Edit Filter...
Delete
Restore Defaults
Apply
Apply and Close
Cancel



  - 鼠标右键+Reference+Project
- 跳转到定义或声明的地方
  - ctrl+鼠标左键
  - F3
- tooltip显示定义：F2
- 打开搜索对话框：ctrl+h
- 全局查找并替换：ctrl+r
- 全局字体放大：ctrl+=
- 全局字体缩小：ctrl+-
- 折叠所有代码：ctrl+shift+小键盘/
- 展开所有代码：ctrl+shift+小键盘*
- 撤销操作：ctrl+z

快捷键自定义方法：Window->Preferences->General->Keys，在搜索框搜索Command快捷键名称或Binding当前绑定快捷键名，搜索后点击该条记录，编辑下方的Binding项，点击应用关闭即可

修改跳转到调用的地方：ctrl+c

修改跳转到定义或声明的地方：ctrl+d

修改全局查找并替换：ctrl+r

修改打开搜索对话框：ctrl+f

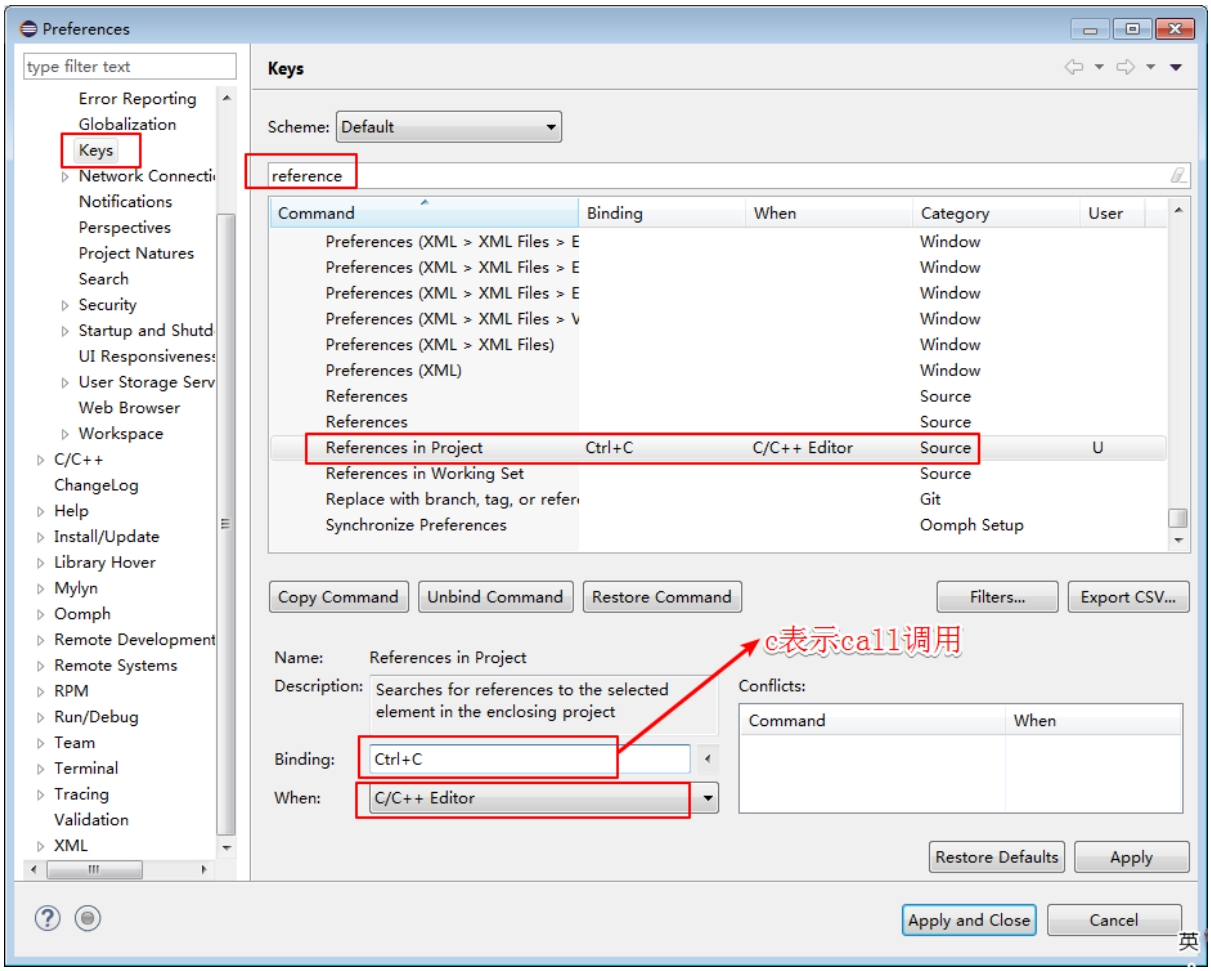
修改代码自动补全：tab

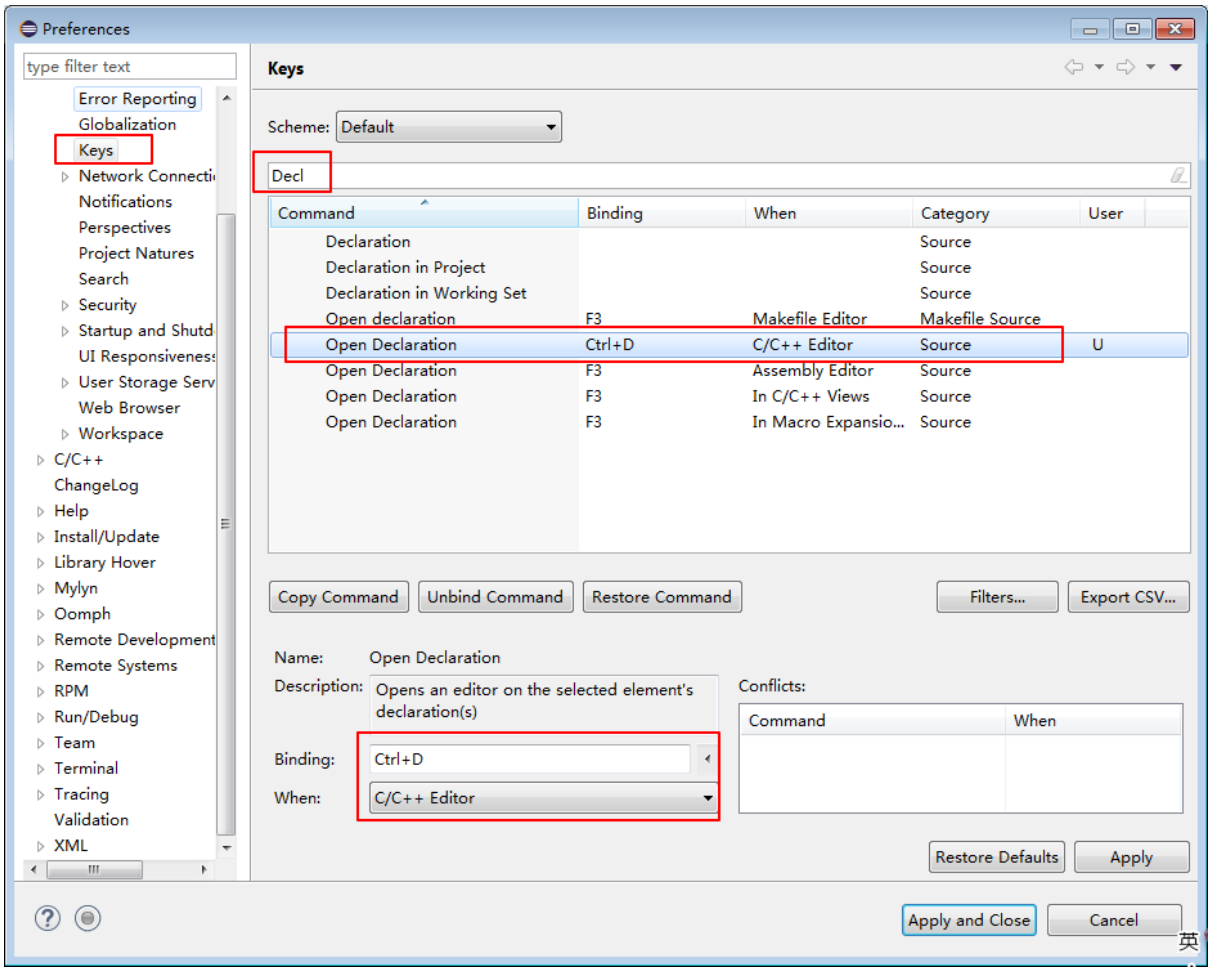
0x03 代码编译

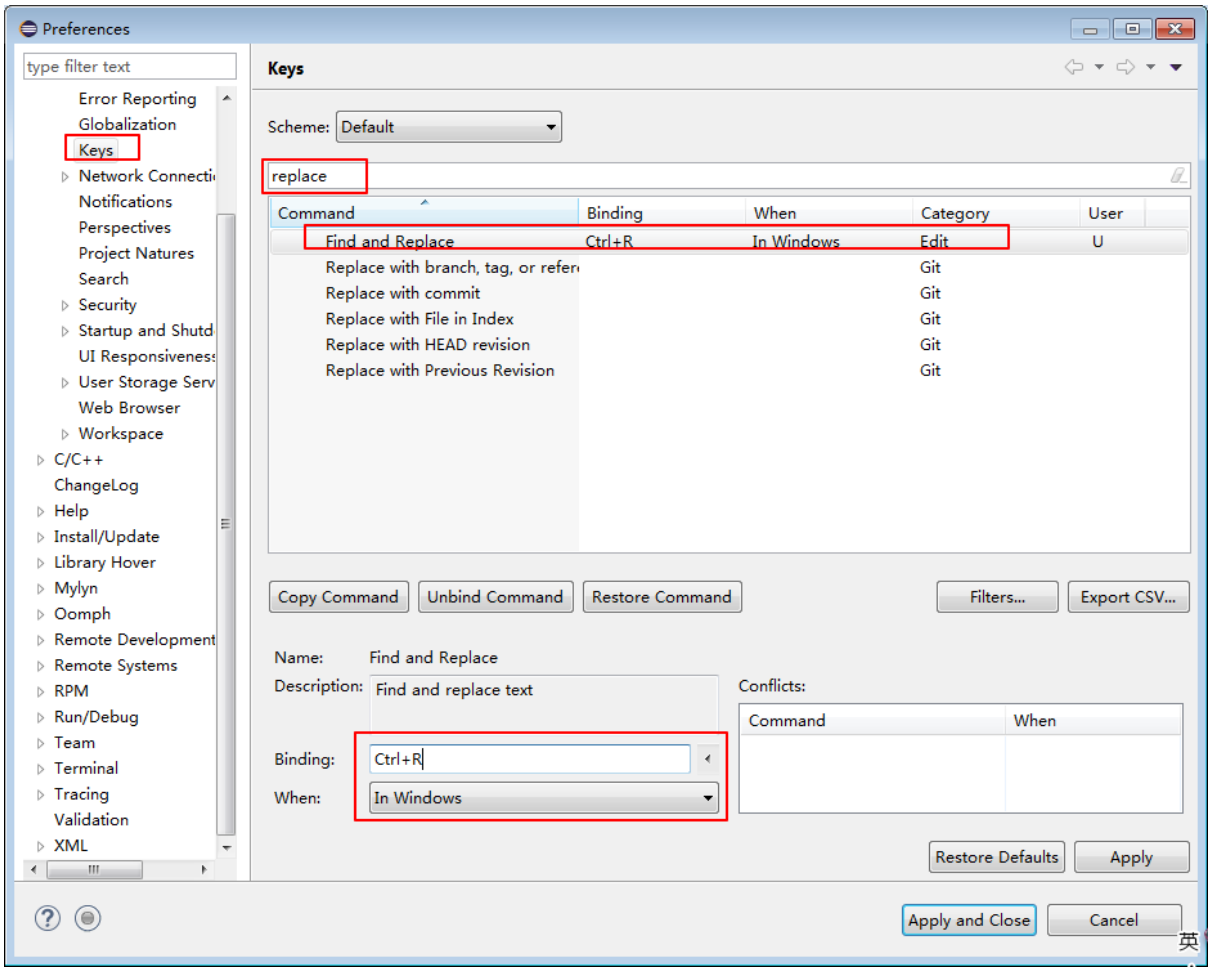
代码编译是在服务器上进行编译，此处之后再补充

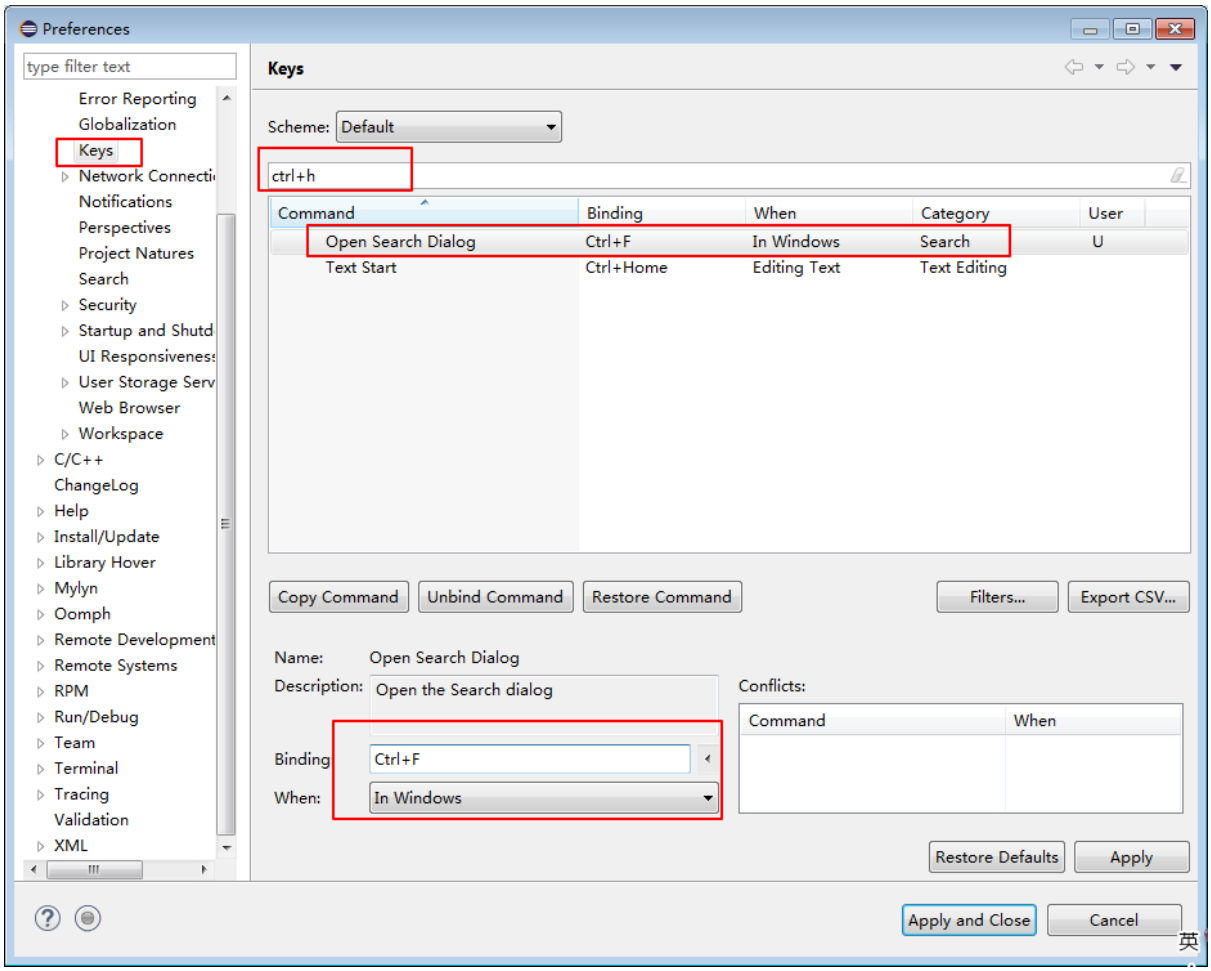
0x04 代码提交

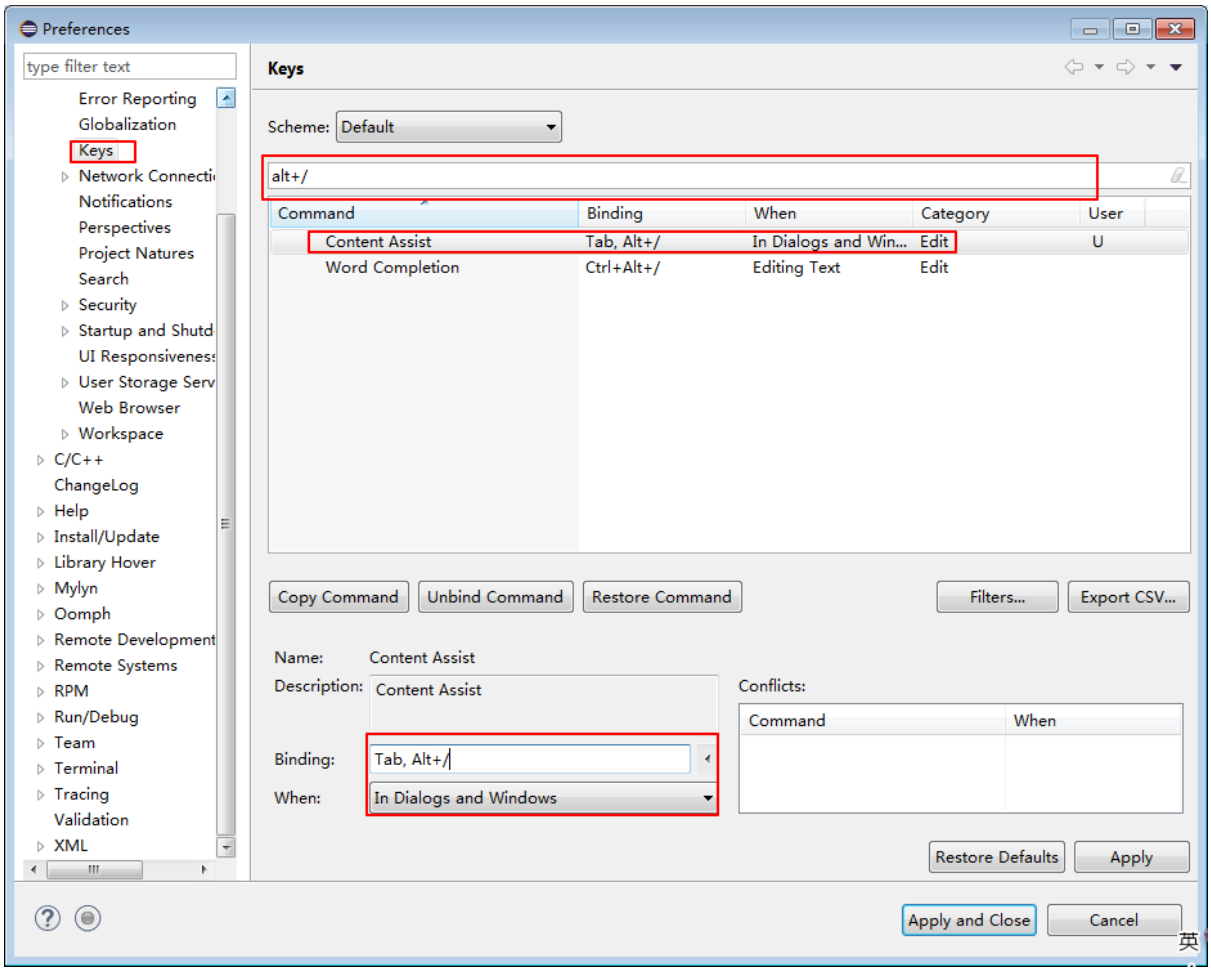
代码提交时使用svn版本控制工具进行提交，可参考svn版本控制


Preferences
type filter text
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Project Natures
Search
Security
Startup and Shutd
UI Responsiveness
User Storage Serv
Web Browser
Workspace
C/C++
ChangeLog
Help
Install/Update
Library Hover
Mylyn
Oomph
Remote Development
Remote Systems
RPM
Run/Debug
Team
Terminal
Tracing
Validation
XML
Keys
Scheme: Default
reference
Command
Binding
When
Category
User
Preferences (XML > XML Files > E
Window
Preferences (XML > XML Files > E
Window
Preferences (XML > XML Files > E
Window
Preferences (XML > XML Files > V
Window
Preferences (XML > XML Files)
Window
Preferences (XML)
Window
References
Source
References
Source
References in Project
Ctrl+C
C/C++ Editor
Source
U
References in Working Set
Source
Replace with branch, tag, or refer
Git
Synchronize Preferences
Oomph Setup
Copy Command
Unbind Command
Restore Command
Filters...
Export CSV...
c表示call调用
Name:
References in Project
Description:
Searches for references to the selected element in the enclosing project
Conflicts:
Command
When
Binding:
Ctrl+C
When:
C/C++ Editor
Restore Defaults
Apply
Apply and Close
Cancel
英


Preferences
type filter text
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Project Natures
Search
Security
Startup and Shutd
UI Responsiveness
User Storage Serv
Web Browser
Workspace
C/C++
ChangeLog
Help
Install/Update
Library Hover
Mylyn
Oomph
Remote Development
Remote Systems
RPM
Run/Debug
Team
Terminal
Tracing
Validation
XML
Keys
Scheme: Default
Decl
Command
Binding
When
Category
User
Declaration
Source
Declaration in Project
Source
Declaration in Working Set
Source
Open declaration
F3
Makefile Editor
Makefile Source
Open Declaration
Ctrl+D
C/C++ Editor
Source
U
Open Declaration
F3
Assembly Editor
Source
Open Declaration
F3
In C/C++ Views
Source
Open Declaration
F3
In Macro Expansio...
Source
Copy Command
Unbind Command
Restore Command
Filters...
Export CSV...
Name: Open Declaration
Description: Opens an editor on the selected element's declaration(s)
Conflicts:
Command
When
Binding: Ctrl+D
When: C/C++ Editor
Restore Defaults
Apply
Apply and Close
Cancel
英


Preferences
type filter text
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Project Natures
Search
Security
Startup and Shutd
UI Responsiveness
User Storage Serv
Web Browser
Workspace
C/C++
ChangeLog
Help
Install/Update
Library Hover
Mylyn
Oomph
Remote Development
Remote Systems
RPM
Run/Debug
Team
Terminal
Tracing
Validation
XML
Keys
Scheme: Default
replace
Command
Binding
When
Category
User
Find and Replace
Ctrl+R
In Windows
Edit
U
Replace with branch, tag, or refer
Git
Replace with commit
Git
Replace with File in Index
Git
Replace with HEAD revision
Git
Replace with Previous Revision
Git
Copy Command
Unbind Command
Restore Command
Filters...
Export CSV...
Name: Find and Replace
Description: Find and replace text
Conflicts:
Command
When
Binding: Ctrl+R
When: In Windows
Restore Defaults
Apply
Apply and Close
Cancel
英


Preferences
type filter text
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Project Natures
Search
Security
Startup and Shutd
UI Responsiveness
User Storage Serv
Web Browser
Workspace
C/C++
ChangeLog
Help
Install/Update
Library Hover
Mylyn
Oomph
Remote Development
Remote Systems
RPM
Run/Debug
Team
Terminal
Tracing
Validation
XML
Keys
Scheme: Default
ctrl+h
Command
Binding
When
Category
User
Open Search Dialog
Ctrl+F
In Windows
Search
U
Text Start
Ctrl+Home
Editing Text
Text Editing
Copy Command
Unbind Command
Restore Command
Filters...
Export CSV...
Name: Open Search Dialog
Description: Open the Search dialog
Conflicts:
Command
When
Binding
Ctrl+F
When:
In Windows
Restore Defaults
Apply
Apply and Close
Cancel
英


Preferences
type filter text
Error Reporting
Globalization
Keys
Network Connecti
Notifications
Perspectives
Project Natures
Search
Security
Startup and Shutd
UI Responsiveness
User Storage Serv
Web Browser
Workspace
C/C++
ChangeLog
Help
Install/Update
Library Hover
Mylyn
Oomph
Remote Development
Remote Systems
RPM
Run/Debug
Team
Terminal
Tracing
Validation
XML
Keys
Scheme: Default
alt+/
Command
Binding
When
Category
User
Content Assist
Tab, Alt+/
In Dialogs and Win...
Edit
U
Word Completion
Ctrl+Alt+/
Editing Text
Edit
Copy Command
Unbind Command
Restore Command
Filters...
Export CSV...
Name: Content Assist
Description: Content Assist
Conflicts:
Command
When
Binding: Tab, Alt+/
When: In Dialogs and Windows
Restore Defaults
Apply
Apply and Close
Cancel
英

定制eclipse

未完待续.....

Source Insight

目录

软件包安装

基本配置

项目工程管理

定制SourceInsight

常见FAQ

PyCharm

软件包安装

基本配置

项目工程管理

定制PyCharm

常见FAQ

1.4 Linux工具集

Linux的哲学思想是：

- 一切皆文件
- 组合目的单一的小程序完成复杂的任务
- 尽量避免跟用户交互
- 使用文本文件保存配置信息
- 提供机制，而非策略
- 缓存为王，缓存是为了平衡各部件之间速度不协调，提高计算机性能的重要组件

参考文档

- linux工具快速教程
- Linux回炉复习系列文章总目录

本系列文档中，所有关于命令的使用介绍都遵循以下逻辑

- 命令功能
- 语法格式

- 格式详解
- 应用实例

1.4.1 目录

Linux概述

Linux是类Unix计算机操作系统的统称

- 参考文档一：Linux发展史及简介
- 参考文档二：Linux发展历史

其发展历史可以简述为

- 1965年，Bell实验室、MIT、GE(通用电气公司)准备开发Multics系统，为了同时支持300个终端访问主机，但是1969年失败了；(刚开始并没有鼠标、键盘、输入设备只有卡片机，因此如果要测试某个程序，则需要将读卡纸插入卡片机，如果有错误，还需要重新来过 Multics：Multiplexed Information and Computing Service)
- 1969年，Ken Thompson(C语言之父)利用汇编语言开发了FIle Server System(Unics，即UNIX的原型)，因为汇编语言对于硬件的依赖性，因此只能针对特定硬件，然而只是为了移植一款太空旅游的游戏
- 1973年，Dennis Ritchie和Ken Thompson发明了C语言，而后写出了UNIX的内核(将B语言改成C语言，由此产生了C语言之父；90%的代码是C语言写的，10%的代码用汇编写的，因此移植时只要修改那10%的代码即可)
- 1977年，Berkeley大学的Bill Joy针对他的机器修改UNIX源码，称为BSD(Berkeley Software Distribution)，Bill Joy是Sun公司的创始人
- 1979年，UNIX发布System V，用于个人计算机
- 1984年，因为UNIX规定：不能对学生提供源码；Tanenbaum老师自己编写兼容于UNIX的Minix，用于教学
- 1984年，Stallman开始GNU(GNU's Not Unix)项目，创办FSF(Free Software Foundation)基金会
  - 其产品有：GCC、Emacs、Bash Shell、GLIBC
  - GNU的软件缺乏一个开放的平台运行，只能在UNIX上运行
  - GNU倡导自由软件，即用户可以对软件做任何修改，甚至再发行，但是始终要挂着GPL的版权；自由软件是可以卖的，但是不能只卖软件，而是卖服务、手册等
- 1985年，为了避免GNU开发的自由软件被其他人用作专利软件，因此创建GPL(General Public License)版权声明
- 1988年，MIT为了开发GUI，成立了XFree86的组织
- 1991年，芬兰赫尔辛基大学的研究生Linus Torvalds基于gcc、bash开发了针对386机器的Linux内核
- 1994年，Torvalds发布Linux-v1.0
- 1996年，Torvalds发布Linux-v2.0，确定了Linux的吉祥物：企鹅

linux发展到现在，已经有诸多发行版本，如下图所示

- 参考文档：linux发行版

在上述所示发行版中，广泛使用的有

- RedHat系列发行版：商业公司维护发行的版本，其包管理方式采用的是基于rpm的yum包管理方式

THE HISTORY OF LINUX

>information

~]$ help -

1971

June 1971 ***Richard Matthew Stallman*** joined MIT Artificial Intelligence Laboratory as a programmer. Sharing codes was done freely that time.

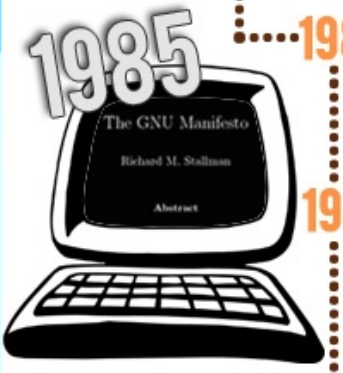
1980

A business model emerged from software compiled to run on different PCs but companies restricted code sharing, copyrighting their software instead.

Years after, Stallman founded the Free Software Foundation believing that software has to be free always.

1985

STALLMAN PUBLISHED THE GNU Manifesto in a bid to create a free OS called GNU that would be compatible with Unix.



1985

ALSO SAW THE CREATION OF MINIX, an OS from scratch for the Intel i386 platform by Professor Andy Tanenbaum.

1989

Stallman released GNU General Public Licence (GPL), the first program independent. All of Stallman's work was under this licence.

GPL Free Software

Free as in Freedom

1989

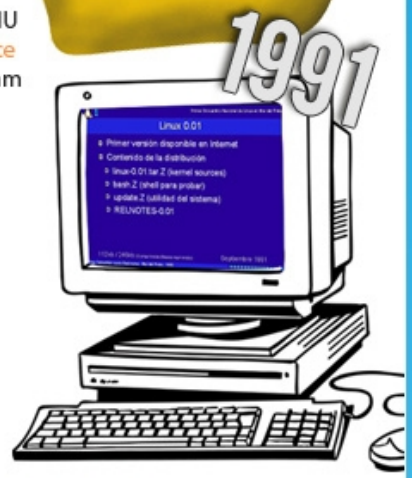
Finnish student Linus Torvalds of University of Helsinki came across Minix and wanted to upgrade it by putting in more features.

UNIVERSITY OF HELSINKI

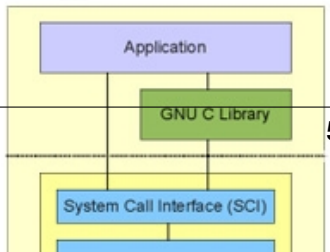
As he was barred from further improving Minix, he wrote his own kernel now known as Linux and released it under GPL.

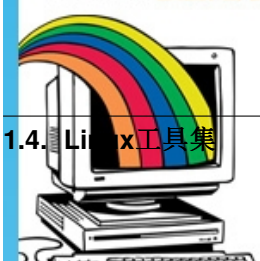
1991

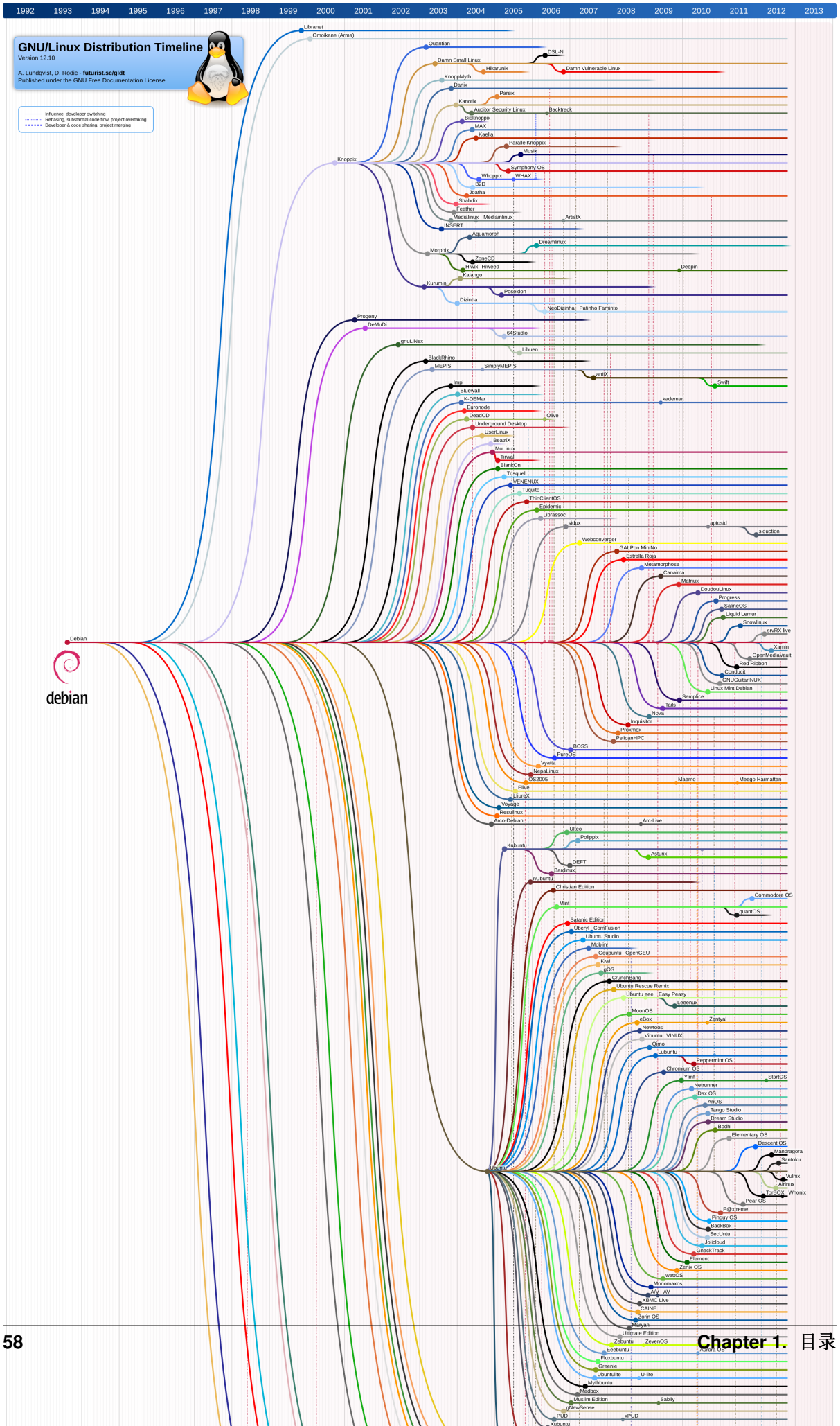
Mid-September 1991 - Linux version 0.01 was released and put on the internet



October 5, 1991 - Linux version 0.02 came about






1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
GNU/Linux Distribution Timeline
Version 12.10
A. Lundqvist, D. Rodic - futurist.se/gldt
Published under the GNU Free Documentation License
Influence, developer switching
Rebasing, substantial code flow, project overtaking
Developer & code sharing, project merging
Debian
debian
Libranet
Omoikane (Arma)
Knoppix
Quantian
DSL-N
Damn Small Linux
Hikarunix
Damn Vulnerable Linux
KnoppMyth
Danix
Kanotix
Parsix
Auditor Security Linux
Backtrack
Bioknoppix
MAX
Kaella
ParallelKnoppix
Musix
Symphony OS
Whoppix
WHAX
B2D
Joatha
Shabdix
Feather
Mediainux
Mediainlinux
ArtistX
INSERT
Aquamorph
DreamInux
Morphix
ZoneCD
Hiwix
Hiweed
Deepin
Kalango
Kurumin
Poseidon
Dizinha
NeoDizinha
Patinho Faminto
Progeny
DeMuDi
64Studio
gnuLiNex
Lihuen
BlackRhino
MEPIS
SimplyMEPIS
antiX
Swift
Impi
Bluewall
K-DEMar
kademar
Euronode
DeadCD
Olive
Underground Desktop
UserLinux
BeatrIX
MoLinux
Tinwal
BlankOn
Trisquel
VENENUX
Tuquito
ThinClientOS
Epidemic
Librassoc
sidux
aptosid
siduction
Webconverger
GALPon MiniNo
Estrella Roja
Metamorphose
Canaima
Matriux
DoudouLinux
Progress
SalineOS
Liquid Lemur
Snowlinux
srvRX live
Xamin
OpenMediaVault
Red Ribbon
Conducit
GNUGuitarINUX
Linux Mint Debian
Semplice
Tails
Nova
Inquisitor
Proxmox
PelicanHPC
BOSS
PureOS
Vyatta
NepaLinux
OS2006
Maemo
Meego Harmattan
Elive
LiiureX
Voyage
Resulinux
Arco-Debian
Arc-Live
Ulteo
Polippix
Kubuntu
Asturix
DEFT
Bardinux
nUbuntu
Christian Edition
Commodore OS
Mint
quantOS
Satanic Edition
Ubery
ComFusion
Ubuntu Studio
Mobilni
Geubuntu
OpenGEU
Kiwi
gOS
CrunchBang
Ubuntu Rescue Remix
Ubuntu eee
Easy Peasy
Leeenux
MoonOS
eBox
Zentyal
Newtoos
Vibuntu
VINUX
Qimo
Lubuntu
Peppermint OS
Chromium OS
Ylmf
StartOS
Netrunner
Dax OS
ArtOS
Tango Studio
Dream Studio
Bodhi
Elementary OS
DescentOS
Mandragora
Santoku
Vulnix
Arinux
TorBOX
Whonix
Pear OS
P@xtreme
Pinguy OS
BackBox
SecUntu
Jolicloud
GnackTrack
Element
Zenix OS
wattOS
Monomaxos
AV
AV
XBMC Live
CAINE
Zorin OS
Ultimate Edition
Zebuntu
ZevenOS
Eeebuntu
Fluxbuntu
Greenie
Ubuntulite
U-lite
Mythbuntu
Madbox
Muslim Edition
Sabily
gNewSense
PUD
xPUD
Xubuntu

  - RHEL(Redhat Enterprise Linux)：也就是Redhat Advance Server收费版本，该发行版稳定性好，适用于服务器
  - CentOS：RHEL的社区克隆免费版本，该发行版稳定性好，适用于服务器
  - Fedora：由原来的Redhat桌面版本发展而来的免费版本，稳定性较差，最好只用于桌面应用
- Debian系列发行版：社区组织维护的发行版本，其包管理方式采用的是基于dpkg的apt-get包管理方式
  - Ubuntu：有应用于服务器的免费版本，也有应用于桌面的免费版本，具体如下
    * 命令规则
      · 前两位数字代表发行时年份的最后两位数字，称为主版本号；主版本号为单数年时是短期支持版本，主版本号为双数年时是长期支持版(LTS)
      · 后两位代表发行的月份，称为副版本号；副版本号要么是4，要么是10；4代表4月份发布的稳定版，10代表10月份发布的测试版版，通常在稳定版中发现的漏洞，或是一些改进方案会放到10月份的测试版本中进行测试
    * 版本支持周期
      · 桌面(Desktop)LTS(Long Term Support)版本至少三年的技术支持
      · 服务器(Server)LTS(Long Term Support)版本至少五年的技术支持
    * 版本发布频率：一年2次；4月份稳定版一次，10月份测试版版一次

关于上述广泛使用的linux发行版，即可以在官网进行下载，又可以在相关镜像站点进行下载

- 阿里aliyun
- 网易163
- 搜狐sohu
- 清华大学
- 中国科技大学
- 东软信息学院

尽管linux的发行版如此众多，但是每种发行版的系统构造大致相同，如下图所示

其中

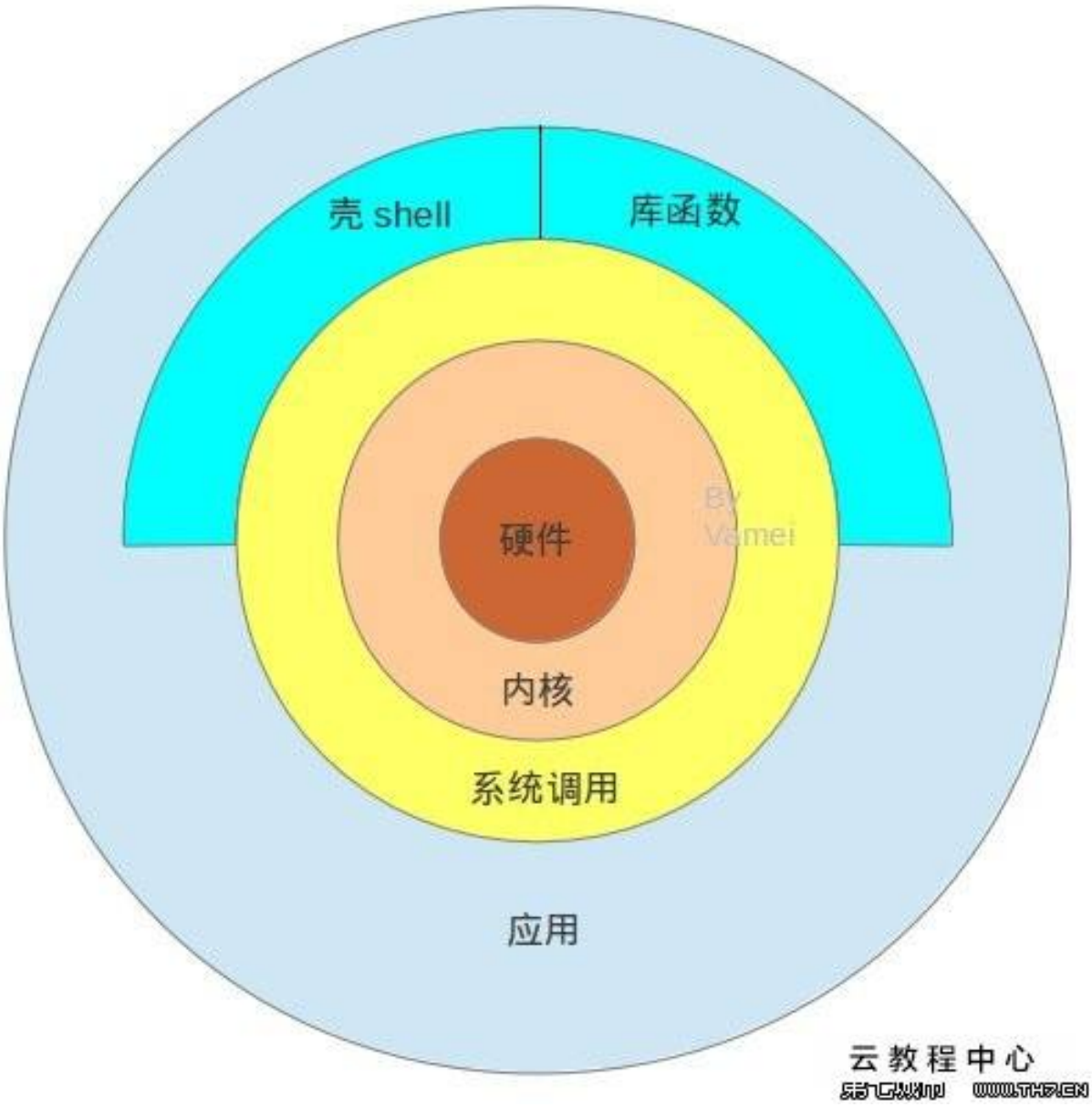
- 每种发行版的内核都是使用linux内核
- 每种发行版提供的应用程序各不相同，包括
  - 编译器：gcc/g++等
  - 文本编辑器：vim/emacs等
  - GUI环境：KED(qt编写)、GNOME(GTK编写)、Unity等
  - CLI环境：shell等
- 库函数是对内核提供的内核态系统调用的用户态封装，给应用程序提供对应的内核入口；应用程序调用库函数，通过对应的系统调用进入内核态，完成对相应硬件设备的操作

对我们用户而言，与linux系统进行交互的方式无非就是两种：GUI环境和CLI环境

其中广泛使用的，也是身为码农必须掌握的交互方式非CLI环境的shell莫属

所以本系列文章将从shell环境、shell命令、shell编程三大章节围绕CLI环境之shell来展开介绍Linux工具集

- **注意**：本系统文章不包括身为c程序猿必须掌握的标准库和系统调用，这两部分的详解可参考：c系统编程之常用类库


壳 shell
库函数
硬件
内核
系统调用
应用
By Vamei
云教程中心
WWW.THEYCEN

shell环境

shell程序==命令解析器

- 命令
- 解析器特性

shell脚本==以某种语法将命令组织起来的由shel程序解析执行的脚本文件

- 命令
- 语法
- 解析器

目录

shell应用程序

- shell本质分类
- shell命令解析：shell解析命令行
- shell相关特性
- shell配置文件：bash环境配置流程

shell命令脚本

shell命令

目录

命令概述

命令汇总

- *type*
- *echo*
- *hash*
- *export*
- *man*
- *whatis*
- *which*
- *whereis*
- *info*

我们将从以下几个方面来了解一下命令的基本概念

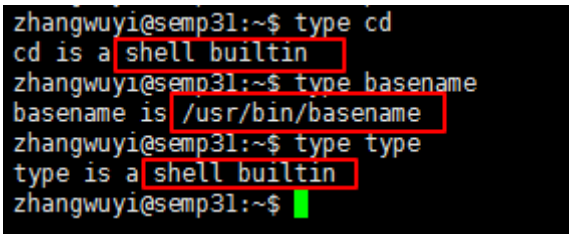
- 命令分类
- 命令执行
- 命令帮助
- 命令语法

0x00 命令分类

在linux中命令可以分为两大类：

- 内部命令(`builtin command`)：在bash中内部实现的命令叫做内建命令，在文件系统上没有对应的可执行文件
- 外部命令(`binary command`)：在文件系统上的某个位置(`/bin`、`/sbin`等)有一个与命令名称对应的可执行文件

我们可以通过`type`命令来判断命令的类型



0x01 命令执行

在linux中命令的执行逻辑是：

- 1.当在shell窗口输入一个命令时，shell进程会读取`hash`查找表，查看该表中是否缓存了输入命令对应可执行文件所在路径，如果缓存了就直接引用该路径，找到可执行文件并执行；如果没有缓存就执行第二步
- 2.shell会通过一个变量`PATH`设定多个路径，当用户输入命令没有在`hash`查找表查找到缓存信息时，shell会自动读取变量`PATH`的值，由左往右到这些路径查找与命令名称相同的可执行文件，然后执行；如果没有找到则会报错，说明没有该命令

```
# 查看hash查找表:用来保存以前执行过命令对应二进制执行文件路径
$hash

# 查看PATH环境变量的值
$echo $PATH    # 在shell语言中变量需要通过$来引用访问

# 将/usr/local/docker/bin路径添加到PATH环境变量中
$export PATH=/usr/local/docker/bin:$PATH    # 该配置只对当前shell进程有效
# 不带任何参数的export只是输出当前shell进程的环境变量的值；如果像上述添加参数，则表示重置指定环境变量的值
```

0x02 命令帮助

在linux终端，面对命令不知道怎么用，或不记得命令的拼写及参数时，我们需要求助于系统的帮助文档；linux系统内置的帮助文档很详细，通常能解决我们的问题，我们需要掌握如何正确的去使用它们

通常我们使用man命令来查看命令的说明文档；使用方法如下：

```
$man COMMAND              # 显示命令的man说明文档
$man 3 COMMAND            # 显示命令的第3类man说明文档
$man -k KeyWord           # 根据部分关键字来查询命令的说明文档；可以使用通配符

# 当man文档中出现乱码的情况时，可以使用export LANG=en命令将语言设置为en
```

在man的帮助手册中，将帮助文档分为了9个类别，有的关键字可能存在多个类别中， 我们就需要指定特定的类别来查看(常用的是分类1和分类3)

- 1：用户可以操作的命令或者是可执行文件
- 2：系统核心可调用的函数与工具等
- 3：一些常用的函数与数据库
- 4：设备文件的说明
- 5：设置文件或者某些文件的格式
- 6：游戏
- 7：惯例与协议等。例如Linux标准文件系统、网络协议、ASCII，码等说明内容
- 8：系统管理员可用的管理条令
- 9：与内核有关的文件

man命令的执行逻辑是：从1~9段(man1,man2.....)依次查找命令关键字第一次出现的帮助文档，先解压该帮助文档然后调用less命令显示帮助文档的文档内容。man文档的存放路径一般是/usr/share/man

所以man默认显示最前面的分类文档，如果一个命令有多个分类文档，我们可以通过whatis命令查看命令存在哪些分类文档。然后再通过man N COMMAND命令查看指定分类文档的信息

```
$whatis COMMAND              # 显示命令所处的man分类页面
$whatis -w "ca*"             # 显示通配符匹配到的命令所处man分类页面

# 如果whatis的数据库尚未生成，可以使用makewhatis手动生成数据库
```

查看man帮助文档时我们可以使用一些快捷键

- 空格键：向文件尾部翻一屏
- b：向文件首部翻一屏
- 回车键：向文件尾部翻一行
- k：向文件首部翻一行

- `ctrl+d`：向文件尾部翻半屏
- `ctrl+u`：向文件首部翻半屏
- `1G`：第一行
- `G`：最后一行

一般的man文档包括NAME、SYNOPSIS、OPTIONS、EXAMPLES、DESCRIPTION几个部分。在SYNOPSIS部分有些特殊字符，它们具有特殊含义：

- `[]`：表示可选的部分
- `{a|b}`：选a或者b，但是必须选择一个
- <>: 表示必不可少的部分
- `...`: 表示同类内容可以出现多个

如果有些命令没有相关的man文档，我们还有其它查看命令帮助的方法

```
$help COMMAND            # 查看内部命令的简要说明文档
$COMMAND --help          # 查看外部命令的简要说明文档
$info COMMAND            # 显示命令的较详细说明文档

# 很多应用程序都会自带文档：/usr/share/doc
# 同样可以查看官方文档
```

有时我们需要查看下命令的相关路径

```
$which COMMAND             # 查看COMMAND的binary文件所在路径
$whereis COMMAND           # 查看COMMAND的binary文件、source文件、man帮助文档所在路径
```

0x03 命令语法

参照man文档的SYNOPSIS字段，命令的格式可以写成：`command options arguments`

- `command`：命令名，可执行文件的文件名
- `options`：命令选项，不同选项代表不同的功能属性
  - 选项可以有多个，多个选项之间必须以空格分隔
  - 短选项：-char（字符）
  - 长选项：–word（单词）
  - 两个短选项可以合并，长选项一般不可以合并
  - 有些选项需要有参数
- `arguments`：命令参数，命令的操作对象
  - 有些命令可以带多个参数，各参数之间需要使用空格隔开

文件管理

目录

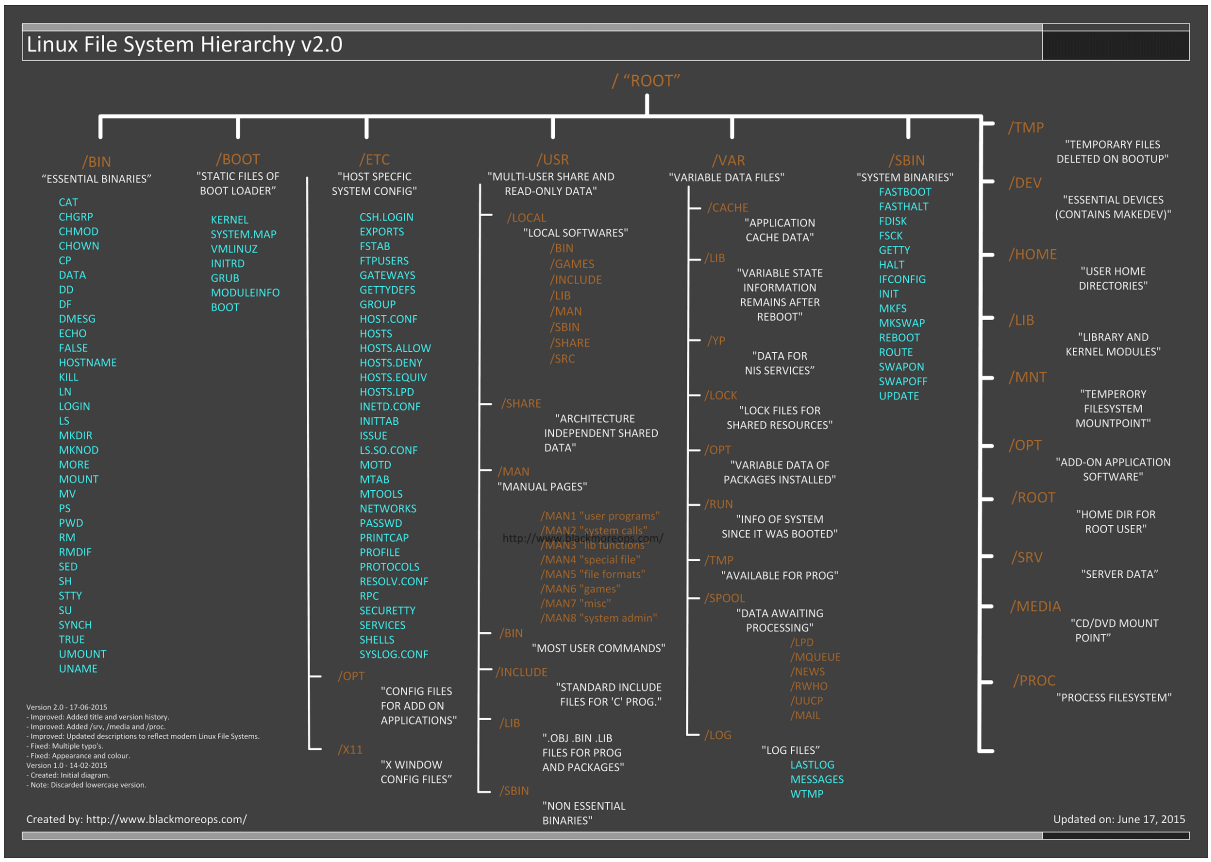
FHS根文件系统

文件系统的本质：实现文件能够按照名称进行存取的内核级别的软件应用程序

FHS根文件系统就是以层次化方式组织所有的文件，根文件系统所在的磁盘分区叫做根分区，也是所有文件的访问入口

所有文件的访问需要预先挂载，挂载的本质就是将该文件所在的分区关联到根分区上某个特定目录下；因此，内核先访问根分区，然后访问与之关联的分区(待访问文件所在分区)，最后访问文件

FHS根文件系统的结构如下



- /：根文件系统的根，系统上所有文件的访问入口
- 存放操作系统自身运行使用的二进制程序
  - /bin: 管理员和普通用户都可以使用
  - /sbin：管理员才能执行的命令
- 存放运行正常功能的二进制程序
  - /usr/bin：管理员和普通用户都可以使用
  - /usr/sbin：管理员才能执行的命令
- 存放第三方软件的二进制程序
  - /usr/local/bin：管理员和普通用户都可以使用
  - /usr/local/sbin：管理员才能执行的命令
- /boot：存放系统引导文件：内核、“ramfs文件“、“bootloader(grub)“
- /dev：存放内核识别的设备文件(设备在内核中会映射成设备文件存放在该目录下,该目录下的设备文件要想实现被访问必须先进行挂载)
  - 该目录的生成机制
    - 1.内核在初始化时通过devtmps(用于在内核初始化时为基本设备创建设备文件的临时文件系统)机制在initramfs(根文件系统所在设备的设备驱动模块)上创建基本设备文件
    - 2.用户空间初始化时通过udev机制在结合/etc/udev/rules.d/*.rules配置文件在/dev目录下创建devtmps尚未创建的设备文件

        * 3.如果还有设备文件没有创建，就使用mknod命令手动创建设备文件
    - cdrom：光盘(CD/DVD)映射形成的便携式设备文件，它是一个符号链接，链接到sr0设备文件
    - sr0：光盘(CD/DVD)映射形成的便携式设备文件
    - zer0：该设备文件存储的全部是0
    - null：存储到该设备文件中的数据都会自动消失
    - console：物理终端控制台，集成在芯片中，系统刚启动时或启动过程中进行交互的终端
    - ttys#：串行终端，使用串口连接的终端，#代表数字
    - tty#：虚拟终端，附加在物理终端上可以任意切换，系统启动完成后进行交互的终端，#代表数字
        * Ctrl+Alt+F1~F6快捷键可以启动或者说切换到虚拟终端tty1~tty6
        * Ctrl+Alt+F7快捷键可以启动图形终端
    - pts/#：伪终端，远程连接或图形界面下打开的命令接口，#代表数字
- /etc：配置文件的集中存放目录
    - fstab：该文件定义了设备文件自动挂载表，系统开机初始化时，会读取该文件根据自动挂载表进行自动挂载设备文件操作。可以通过编辑该文件实现开机自动挂载设备；该文件中六段的意义如下：
        * 1、要挂载的设备：设备的描述形式===设备文件路径、LABEL=""、UUID=""
        * 2、挂载点：有的文件系统没有挂载点，swap就没有挂载点，它的挂载点就是swap
        * 3、文件系统类型：xfs、swap、ext4
        * 4、挂载选项：多个选项间使用逗号分隔，默认是defaults
        * 5、备份频率：0-从不备份、1-每日备份、2-每隔一天备份
        * 6、开机自检次序：1-首先自检，通常只能被/使用、2-根自检完后再自检、0-从不自检
    - services：名称解析库，实现端口和服务名称之间进行转换
    - mtab：存放当前系统所有的设备挂载信息，mount命令会自动修改该文件
    - init.d：目录，存放系统开机初始化脚本文件
    - ld.so.conf：存放程序编译源代码过程中链接(ld)时所要查找动态格式共享对象库时所要查找的路径
    - ld.so.conf.d：目录，存放动态链接库查找路径的所有配置文件
    - sysconfig：存放系统级别的应用信息
- /home：普通用户，默认在/home下有一个与其名称同名目录，作为用户的家目录
- /root：管理员的家目录
- /lib：存放32位库文件
- /lib64：存放64位库文件
- /media：专用挂载位置，通常用来挂载便携式设备
- /mnt：专用挂载位置，通常用来挂载额外的存储设备
- /misc：杂项，备用目录
- /opt：可选目录，但通常用来安装第三方软件
- /proc：所显示的文件都不是文件，伪文件系统，保存运行中的内核参数的映射，不能使用vim等编辑器打开；该目录下的绝大数文件都没有写权限，即使是管理员也无法编辑修改大多数文件；linux将内核中所有进程的参数通过/proc伪文件系统目录向用户空间提供访问查看的入口。每一个进程会以自己的PID号为目录名创建一个目录文件来保持自己的相关信息

  - `version`：存放当前系统正在运行内核的版本
  - `partitions`：存放当前系统所有挂载磁盘的分区信息
  - `meminfo`：存放当前系统的内存信息
  - `filesystems`；存放当前系统内核识别出的文件系统的类型信息
  - `mounts`：存放当前系统所有的设备挂载信息，内核维持并修改该文件的信息
  - `vm/swappiness`：该文件定义了Linux的内存使用机制
  - `sys`：该目录下的文件有很多是提供写权限的，即提供了修改内核参数的入口
  - `cmdline`：内核启动时传递给内核的参数。每个进程目录下都有一个该文件用来传递参数给内核
- `/sys`：伪文件系统，系统级别的用于配置硬件设备相关的参数
- `/srv`：为服务提供数据存放位置
- `/tmp`：临时文件系统：默认存放30天
- `/usr`：存放应用程序的相关文件
  - `shared`: 存放应用程序的说明帮助文档
  - `include`：存放应用程序的头文件
- `/var`：存放经常发生变化的文件
  - `/var/log`：存放日志文件
  - `/var/lock`：存放锁文件
  - `/var/cache`：存放缓存文件

文件属性模型

命令汇总：

- *ls*
- *stat*
- *file*
- *chown*
- *chgrp*
- *uamsk*
- *chmod*
- *getfacl*
- *setfacl*

使用`ls -l /path/to/FileName`命令可以调用目录的可执行权限查看指定文件的属性模型

`ls`命令还有其它选项：

```
$ls -t        # 按照时间戳排序显示文件
$ls -rt       # 按照时间戳逆序显示文件
```

该属性模型依次可划分为以下几段

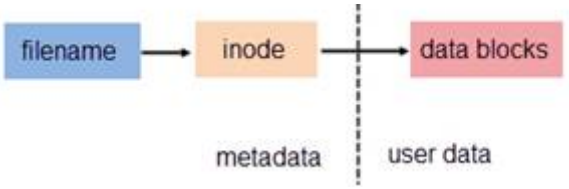
- 文件类型
- 文件权限模型
- 文件硬链接次数

```
[root@localhost ~]# ls -lh /etc
total 1.4M
drwxr-xr-x.  3 root root  101 Apr  8 03:45 abrt
-rw-r--r--.  1 root root   16 Apr  8 04:00 adjtime
-rw-r--r--.  1 root root 1.5K Jun  7  2013 aliases
-rw-r--r--.  1 root root  12K Apr  8 04:03 aliases.db
drwxr-xr-x.  2 root root   51 Apr  8 03:46 alsa
drwxr-xr-x.  2 root root 4.0K Apr  8 03:54 alternatives
drwxr-xr-x.  3 root root   23 Apr  8 03:45 amanda
-rw-------.  1 root root  541 Mar 31  2016 anacrontab
-rw-r--r--.  1 root root   55 Nov  5 00:29 asound.conf
-rw-r--r--.  1 root root    1 Nov  6 02:16 at.deny
drwxr-xr-x.  2 root root   32 Apr  8 03:49 at-spi2
drwxr-x---.  3 root root   43 Apr  8 03:46 audisp
drwxr-x---.  3 root root   83 Apr  8 04:03 audit
-rw-r--r--.  1 root root  14K Nov  6 02:19 autofs.conf
-rw-------.  1 root root  232 Nov  6 02:19 autofs_ldap_auth.conf
-rw-r--r--.  1 root root  795 Nov  6 02:19 auto.master
drwxr-xr-x.  2 root root    6 Nov  6 02:19 auto.master.d
-rw-r--r--.  1 root root  524 Nov  6 02:19 auto.misc
-rwxr-xr-x.  1 root root 1.3K Nov  6 02:19 auto.net
-rwxr-xr-x.  1 root root  687 Nov  6 02:19 auto.smb
drwxr-xr-x.  4 root root   71 Apr  8 03:52 avahi
drwxr-xr-x.  2 root root 4.0K Apr  8 03:53 bash_completion.d
-rw-r--r--.  1 root root 2.8K Nov  6 01:19 bashrc
drwxr-xr-x.  2 root root    6 Nov  7 03:48 binfmt.d
drwxr-xr-x.  2 root root   23 Apr  8 03:45 bluetooth
drwxr-xr-x.  2 root root  12K Apr  8 03:47 brltty
-rw-r--r--.  1 root root  22K Nov  6 01:31 brltty.conf
-rw-r--r--.  1 root root   38 Nov 30 02:12 centos-release
-rw-r--r--.  1 root root   51 Nov 30 02:12 centos-release-upstream
```

- 文件所有者模型
- 文件大小
- 文件时间戳
- 文件名

我们知道文件都有文件名与数据，这在Linux上被分成两个部分

- 用户数据(`user data`)：即文件数据块(data block)，数据块是记录文件真实内容的地方，存放在磁盘的有效数据区
- 元数据(`metadata`)：即文件的附加属性，如inode索引号，文件名，文件时间戳，权限模型，大小，类型，所有者等信息，这些信息存放在磁盘的元数据区。在Linux中，元数据中的inode号(inode是文件元数据的一部分但其并不包含文件名，inode号即索引节点号)才是文件的唯一标识而非文件名。文件名仅是为了方便人们的记忆和使用，系统或程序通过inode号寻找正确的文件数据块。当磁盘块对应的inode没有被任何一个文件名引用(即硬链接数为0)，系统认为该磁盘块处于未使用空闲状态，将会标记该磁盘块为空闲从而回收该磁盘空间。其逻辑图如下所示



电脑能够识别的是数字，存储在元数据区的不是文件名而是索引结点号`index node`；计算机通过名称解析(`name resoling`)将识别的索引结点号解析成人类易读的文件名；将文件名转化成索引号存储，便于计算机识别。

使用`ls -l`命令查看的文件所有属性都存放在文件的metadata元数据区，我们可以通过`stat`命令来查看一个指定文件的所有元数据属性

```
zhangwuyi@semp31:~$ stat str.c
  File: ‘ str.c’
  Size: 1115            Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d      Inode: 45101927    Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1028/zhangwuyi)   Gid: ( 1028/zhangwuyi)
Access: 2017-09-05 17:55:42.311524869 +0800
Modify: 2017-08-24 15:49:47.065501320 +0800
Change: 2017-08-24 15:49:47.113501321 +0800
 Birth: -
```

0x00 文件类型

linux中一切皆文件，其文件类型有：

文件类型	标识符	相关说明
普通文件	-/f	文本文件
目录文件	d	路径映射
硬链接文件	-/d/f	文件名别名
软链接文件	l	快捷方式
字符设备文件	c	线性串行设备
块设备文件	b	随机并行设备
命名管道文件	p/fi/fo	实现本机进程间通信
套接字文件	s	实现跨机进程间通信

我们可以通过`file`命令查看指定文件的文件类型

```
zhangwuyi@semp31:~$ file str.c
str.c: C source, ASCII text
zhangwuyi@semp31:~$
```

普通文件基本上都是文本文件，在磁盘块中存放的是文本内容

目录文件是一种特殊的文件，它是一种**路径映射**。它在磁盘块中存放的数据是该目录下所有文件的文件名以及对应inode号构成的索引表。系统就是通过该路径映射找到文本文件并对其进行读写操作

linux的文件系统是一种遵循`FHS`标准的层次化根文件系统,每次操作文件时都需要从根开始进行索引查找，这大大降低了系统文件访问效率。为了提高效率就出现了`buffer`缓冲元数据和`cache`缓存文本数据。除此之外为两个文件之间建立链接也是一种提高文件访问效率的方法。链接为Linux 系统解决了文件的共享使用，还带来了隐藏文件路径、增加权限安全及节省存储等好处

硬链接文件具有以下特点：

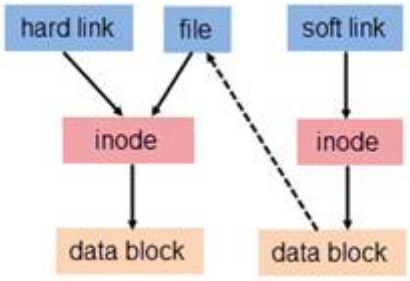
- 硬链接文件和源文件不能跨分区，跨文件系统
- 硬链接文件和源文件是指向同一个inode的不同文件名，即硬链接文件是源文件的一个别名
- 不能对目录创建硬链接，避免循环引用
- 硬链接需要对文件本身进行引用，所以它会改变源文件被链接的次数

软链接文件也可叫做符号链接,具有以下特点：

- 符号链接文件和源文件是可以跨分区
- 符号链接文件和源文件是指向不同inode的不同文件名，可以理解为快捷方式
- 符号链接文件的inode号对应的用户数据块存放的内容是被链接源文件的路径(即源文件文件名)，使用`ls -l`命令显示的符号链接的大小指的是路径字符串的长度
- 可以对目录创建符号链接

- 符号链接只是引用文件路径，不会对文件本身进行引用，所以它不会改变源文件被链接的次数

硬链接和软链接的访问机制如图所示



删除软链接不会对源文件造成影响，因为它删除的只是源文件的路径映射；但是如果删除的硬链接是最后一个，那么对应的inode区就没有文件名引用，inode对应的磁盘空间将会被回收

关于硬链接和软链接的详解可参考理解Linux的硬链接与软链接

连接到系统上的设备都会被内核通过设备驱动识别并映射成/dev目录下的一个设备文件，通过该设备文件对设备进行访问等操作。设备文件可以分为：

- **字符设备**：它是线性串行设备，遵循时序逻辑，一次存取一个字符，例如：键盘等
- **块设备**：它是随机并行设备，不遵循时序，多线程进行，例如：硬盘等

0x01 文件权限模型

权限模型是用来实现系统资源的分配，该模型的基本思想就是在文件系统的基础上为每一个文件标注其所有者及其访问权限。权限模型的作用对象是文件

创建文件后，所有者模型中的每类用户都会有自己的rwx权限模型

ls -l命令显示结果rwxr-xr-x

- 前三位rwx对应文件所有者owner属主的权限
- 中间三位r-x对应文件所有者group属组的权限
- 最后三位r-x对应文件所有者other其它用户的权限

由此可将权限模型抽象成：rwx

- 对文件而言
  - **r(read)**：可以使用内容查看类的命令来显示其相关内容
  - **w(write)**：可以使用编辑器修改其内容
  - **x(execute)**：可以将其发起一个进程
- 对目录而言
  - **r(read)**：可以使用ls命令查看目录内的文件信息
  - **w(write)**：可以创建、删除文件
  - **x(execute)**：可以使用ls -l命令来查看目录内容的文件信息，并且可以使用cd命令切换此目录为工作目录

需要注意的是：

- 用户不拥有某位权限，则使用-占位：r-x表示读和执行的权限、r--表示只读权限、rw-表示读写权限
- rwx权限模型可以使用8机制来表示：r:4、w:2、x:1、-:0
  - r-x可以用5来表示

  - rw-可以用6来表示
  - rwxr-xr-x可以用755来表示
- 所有链接文件的权限都是777并且无法被改动

通常我们在创建一个文件时，并没有特意去指定文件的权限模型，但是创建好的文件同样拥有自己的权限模型，这是因为umask机制：

- 创建普通文件时文件的默认权限模型是666-umask确保普通文件默认不允许出现执行权限，如果出现则在八进制的基础上加1
- 创建目录文件时目录的默认权限模型是777-umask确保目录文件默认应该具有执行权限，如果没有执行权限也可以
- 创建链接文件时链接文件的默认权限都是777并且无法被改动

不同情况下umask的值是不一样的：

- root用户的umask=0022
- 普通用户如果用户名和基本组名一致则umask=0002，否则umask=0022

uamsk的值可以通过umask命令指定数值进行修改，但是此次修改只对当前进程有效，要想永久有效，需要放在配置文件中

```
# 查看umask的值
$umask

# 设置umask的值
$umask 0023
```

对于已经创建好的文件，我们可以通过chmod命令来修改文件的权限模型

```
# 方法一：使用八进制的形式一次性操作三类用户的权限
$chmod 770 ./1.txt
$chmod -R 6 /u        # 当八进制权限模型不足时，默认以0补全，即006；递归修改目录以及子文件的权限模型

# 方法二：基于+/=/-，使用x/w/r，来操作指定用户(u,g,o,a)的权限
$chmod u=rx /u    # 属主的权限是r-x；如果是u=，则表示没有权限
$chmod g-x /u     # 属组去掉x权限
$chmod o+w /u     # 其它用户添加w权限
$chmod a+r /u     # 所有用户添加r权限，此时可以直接写成+r
$chmod -R u+x,g=,o= /u # 同时指定多个用户权限时使用逗号隔开；递归修改目录以及子文件的权限模型

# 方法三：参照其他文件的权限模型修改当前文件权限模型
$chmod --reference=./2.txt /u
$chmod -R --reference=./2.txt /u   # 递归修改目录以及子文件的权限模型
```

接下来我们来说下权限模型的访问应用法则：

- 当用户发起一个进程访问一个文件时，首先来判定发起进程的用户跟文件的属主是否一致，如果一致则应用文件属主的权限；
- 如果不是，则判定用户所属属组中的一个（基本组或者附加组）跟文件的属组是否一致，一致则应用文件属组的权限；
- 如果不是，则应用其他用户的权限

上述所说的权限只是基本权限模型，在linux中存在以下特殊权限模型

- suid：任何用户执行可执行文件发起进程时，不再以用户自己的身份当作进程的属主，而是以可执行文件文件的属主当作进程的属主
  - suid表现为可执行文件属主权限执行位上的s(x)或S(-)
  - suid只对可执行文件有意义

- `sgid`：具有sgid的目录，用户在此创建文件时，新建文件的属组不再是用户的基本组，而是目录的属组
  - sgid表现为目录文件属组权限执行位上的`s(x)`或`S(-)`
  - sgid只对目录文件有意义
- `sticky`：对于公共可写的目录，用户可创建文件，可以删除自己的文件，但无法删除别的用户的文件,该机制为sticky粘滞位
  - sticky表现为目录文件其他用户权限执行位上`t(x)`或`T(-)`
  - sticky只对目录文件有意义

`suid、sgid、sticky`刚好可以类似于`r、w、x`组成一个三位8进制的特殊权限：

- suid为4
- sgid为2
- sticky为1

我们同样可以通过`chmod`命令来修改特殊权限模型

```
# 方法一：使用八进制的形式一次性操作所有特殊权限位
$chmod 4554 /u   # 假设可执行文件原来的权限为455；给可执行文件属主添加suid权限
$chmod 2552 /u   # 假设目录文件原来的权限为255；给目录文件属组添加sgid权限
$chmod 1551 /u   # 假设目录文件原来的权限为155；给目录文件其它用户添加sticky权限

# 方法二：基于+/-，使用s/t，来操作指定用户(u,g,o)的权限
$chmod u+s /u   # 给可执行文件属主添加suid权限
$chmod g+s /u   # 给目录文件属组添加sgid权限
$chmod o+t /u   # 给目录文件其它用户添加sticky权限
```

除了基本权限模型和特殊权限模型外，linux中还存在另外一种权限模型：访问控制列表`facl`

- 普通用户无法安全地将某文件授权给其他用户访问，此时我们在文件原有权限模型之上附加另一层权限控制机制，保存至文件扩展属性信息中，使普通用户能够安全的将自己的文件授权给指定用户进行访问等操作
- 它表现为9位基本权限模型后面的+号，一旦使用`ls -l`命令查看文件权限模型中出现+号说明该文件具有额外的访问控制列表`facl`权限
- facl适用于普通用户指定权限

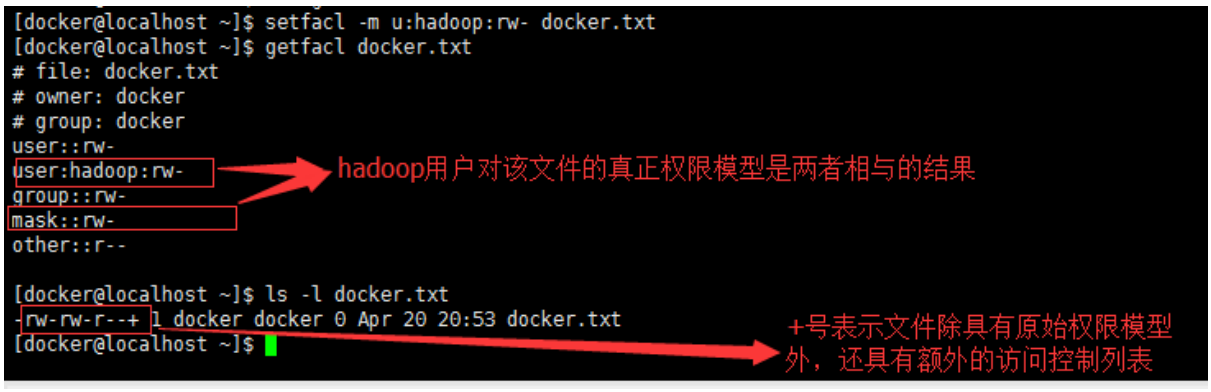
我们可以通过`getfacl`命令查看指定文件的访问控制列表，其输出格式为

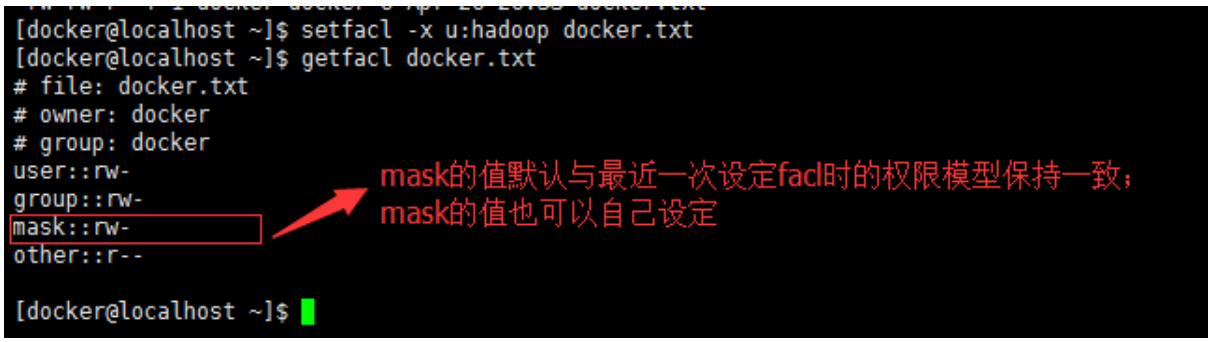
- `user::rw-`：冒号将该字段分为3段，即用户、用户名、对应权限。空格表示该文件的属主
- `group::rw-`：冒号将该字段分为3段，即组、组名、对应权限。空格表示该文件的属组
- `other::r--`：冒号将该字段分为3段，即其他用户、用户名、对应权限。空格表示该文件的其他用户

```
[docker@localhost ~]$ getfacl docker.txt
# file: docker.txt
# owner: docker
# group: docker
user::rw-
group::rw-
other::r--

[docker@localhost ~]$
```

上图是没有设定facl时文件的原有权限模型，我们可以通过`setfacl`命令来设定和取消文件的facl

```
# 设定facl权限
$setfacl -m u:hadoop:rw- /u   # 设定文件指定用户的权限模型
$setfacl -m g:hadoop:rw- /u   # 设定文件指定组的权限模型
$setfacl -m m::rw- /u         # 设定文件的mask权限模型

# 取消facl权限
$setfacl -x u:hadoop /u   # 取消文件指定用户的权限模型
$setfacl -x g:hadoop /u   # 取消文件指定组的权限模型
$setfacl -x m: /u         # 取消文件的mask权限模型

#注意
#1、指定用户和组的真正权限模型是设定值与mask值相与的结果
#2、使用setfacl命令设定facl时只对当前文件有效；如果当前文件是目录文件，且想对目录中的文件也设定facl，则需要使用-R选项实现递归设定facl
```





类似于基本权限模型，访问控制列表facl的应用法则是：

- 先匹配原始用户与文件属主
- 然后匹配facl设定的用户与文件属主
- 然后匹配原始属组与文件属组
- 然后匹配facl设定的属组与文件属组
- 最后匹配原始其他用户与文件其他用户

0x02 文件硬链接次数

关于文件硬链接次数可参考链接文件

也可参考理解Linux的硬链接与软链接

0x03 文件所有者模型

文件的所有者模型的本质就是用户模型，在linux中用户可分为owner属主用户、group属组用户、other其它用户，它们具有以下特点：

- 属主可以属于基本组或附加组，属组包含属主
- other用户是除group组内所有用户之外的其它所有用户
- 每类用户都有其对应的权限模型

每个文件都有其对应的属主和属组，当我们创建文件时，系统会默认将当前用户作为文件的属主，将当前用户的基本组或者附加组作为该文件的属组

我们可以通过chown和chgrp命令来修改文件的属主和属组属性

chown可以修改文件的属主和属组

```
# 将文件的属主改为root
$chown root ./1.txt

# 将文件的属主改为root、属组改为staff
$chown root:staff ./1.txt

# 将文件的属组改为staff
$chown :staff ./1.txt
$chown .staff ./1.txt

# 按照参考文件的所有者模型修改指定文件的所有者模型
$chown --reference=./2.txt ./1.txt

# 当改变一个目录的属主或属组时，默认是不会改变内部文件的属主或属组
# 如果要想改变其内部文件可以使用-R选择，实现递归改变内部文件的属主或属组
$chown -R root /u
```

chgrp用来修改文件的属组

```
# 修改文件的属组为staff
$chgrp staff ./1.txt

# 将目录以及其子文件的属组都修改为staff
$chgrp -R staff /u

# 按照参考文件的所有者模型修改指定文件的所有者模型
$chgrp -R --reference=./2.txt /u
```

0x04 文件大小

ls命令默认显示的文件大小的单位都是字节，当文件过大时不便于人们读取；我们在使用ls命令时可以加上-h选项将文件大小转换为人们易于理解阅读的方式：文件大小以**M**、**G**的方式显示，没有后缀的是以B字节为单位的

0x05 文件时间戳

一个文件的元数据区中存放文件的三种时间戳

- access time(atime)：访问时间
- modify time(mtime)：修改时间(修改用户数据(user data)的时间)
- change time(ctime)：改变时间(修改元数据(metadata)的时间)

```
zhangwuyi@semp31:~$ stat str.c
  File: ' str.c'
  Size: 1115            Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d      Inode: 45101927    Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1028/zhangwuyi)   Gid: ( 1028/zhangwuyi)
Access: 2017-09-05 17:55:42.311524869 +0800
Modify: 2017-08-24 15:49:47.065501320 +0800
Change: 2017-08-24 15:49:47.113501321 +0800
 Birth: -
```

我们可以通过`stat`命令查看这三种时间戳

ls命令显示的文件时间戳只是文件的**ctime改变时间**

0x06 文件名

ls命令默认不能显示以`.`开头的文件，这些文件被称为隐藏文件，如果想要查看这类文件，需要在使用ls命令时加上`-a`选项

文件基本操作

首先，我们基于文件系统逻辑组成部分对常见的文件操作进行深入理解：

- 创建文件：文件名为/etc/rc.d/init.d
  - 先在磁盘上找一段空闲的磁盘块，在inode区建立其磁盘块地址与新建inode号的对应表
  - 然后找到存放rc.d目录内容(该内容是一个文件名与inode号组成的索引表)的磁盘块
  - 在该目录索引表中添加init.d文件名和新建inode号的对应关系
- 不彻底删除文件
  - 删除inode区的文本内容存放块地址与inode号的对应关系
  - 删除文件上一级目录文件(索引表)中的待删除文件的文件名与inode号的对应关系
  - 文件本身存储在数据区的内容不做改变。
- 粉碎文件
  - 删除inode区的文本内容存放块地址与inode号的对应关系
  - 删除文件上一级目录文件(索引表)中的待删除文件的文件名与inode号的对应关系
  - 将文件本身存储在数据区块中的数据用其他数据覆盖填充，破坏原来的数据。
- 同分区下的复制文件：将/etc/fstab复制到/etc/Fstab
  - 在inode区新建一个inode号与存储/etc/fstab内容的块地址相对应
  - 在etc目录文件(索引表)中添加Fstab文件名与新建inode号的对应关系
- 同分区下的移动文件：将/etc/fstab移动到/etc/Fstab
  - 先删除etc目录索引表中fstab文件名与inode号(该inode号与存放fstab文件内容的磁盘块地址相对应)的对应关系
  - 然后在home目录索引表中添加该inode号与Fstab文件名的对应关系
- 跨分区复制文件：将/etc/fstab复制到/home/Fstab
  - 先在目标分区上划分一段空闲的磁盘块，建立好Fstab文件名与inode号，inode号与块地址的对应关系
  - 然后访问/etc/fstab文件，将其内容复制到之前划分好的磁盘块中

- 跨分区移动文件：将/etc/fstab移动到/home/Fstab
  - 先在目标分区上划分一段空闲的磁盘块，建立好Fstab文件名与inode号，inode号与块地址的对应关系
  - 然后访问/etc/fstab文件，将其内容复制到之前划分好的磁盘块中
  - 最后将fstab文件名与inode号,inode号与磁盘块地址之间的对应关系删除

综上所述，对文件的管理基本上是对inode区以及目录索引表的操作，磁盘块中存放的文本数据要么不动，要么就是对其进行数据覆盖。

注意：以上所有关于删除inode号与磁盘块地址的对应关系可以直接通过将inode位图中该inode号对应的位置0实现，表示该inode号是空闲的，可以随意对其对应的磁盘块进行数据覆盖；每新建一个inode号与磁盘块地址的对应关系时，需要将inode位图中该inode号对应的位置1，表示该inode号已被占用。

目录

普通文件

命令汇总

- *touch*
- *rm*
- *mv*
- *cp*
- *find*
- *locate*
- *ln*
- *gzip*
- *gunzip*
- *bzip2*
- *bunzip2*
- *xz*
- *unxz*
- *zip*
- *unzip*

普通文件的基本操作有

- 文件创建
- 文件删除
- 文件移动
- 文件复制
- 文件查找
- 文件链接
- 文件压缩
- 文件解压

0x00 文件创建

创建文件有以下方式

- 使用touch命令
- 使用文本编辑器

touch本来是用来修改已存在文件的时间戳，当文件不存在时可以用来创建空文件

```
# 不使用任何选项且文件不存在时，创建空文件
# 文件的atime/ctime/mtime都为当前时间
# 文件的权限模型为默认权限模型，属主和属组分别是当前用户和当前用户的基本组
$touch 1.txt

# 不使用任何选项但是文件存在时，将指定文件的atime/ctime/mtime都改为当前时间
$touch 1.txt

# 将atime/mtime改为指定时间，ctime自动会改为当前时间
$touch -t 201803150949.22 /u    # 时间戳格式为YYMMDDhhmm.ss

# 仅修改文件的atime，同时ctime会自动修改为当前时间
$touch -at 201803150953.55 /u   # 时间戳格式为YYMMDDhhmm.ss

# 仅修改文件的mtime，同时ctime会自动修改为当前时间
$touch -mt 201803150953.22 /u   # 时间戳格式为YYMMDDhhmm.ss
```

除此之外，还可以通过文本编辑器vim、nano来创建文件

```
# 当vim、nano后面的文件名指定的文件不存在时就为创建新文件，否则为编辑该文件
$vim 1.txt
$nano 1.txt
```

0x01 文件删除

我们可以使用rm命令来实现文件的删除

```
$rm 1.txt      # 删除指定文件
$rm -f 1.txt   # 强制删除指定文件
$rm *.c        # 使用文件名通配机制，删除所有以.c结尾的文件
```

注意：指定文件名时可以使用文件名通配机制

0x02 文件移动

我们可以使用mv命令来实现文件的移动，也可以理解为文件剪切粘贴

```
$mv SRC DEST
$mv 1.txt /tmp    # 将1.txt剪切到/tmp目录下
# 假如SRC是一个文件
#      如果目标是一个文件且目标存在：覆盖
#      如果目标文件不存在：创建新文件，也可以理解为修改文件名
#      如果目标存在，且是个目录：剪切源至目标目录中，并保持原名

$mv SRC... DEST
$mv ./*.c /tmp  # 将当前目录下的所有.c文件剪切到/tmp目录下
# 假如SRC有多个文件：
#     如果目标存在，且是一个文件：移动无法进行
#     如果目标存在，且是一个目录：剪切各文件至目标目录中，并保持原名
```

(continues on next page)

(continued from previous page)

```
#     如果目标不存在：剪切无法进行
# 如果SRC只有一个且是目录：对目录实现剪切时不需要使用-r选项
#     如果目标是一个文件且目标存在：移动失败
#     如果目标文件不存在：创建新目录，实现整个目录的剪切
#     如果目标存在，且是个目录：剪切源目录到目标目录中，并保持原名

# mv命令的常用选项：
#     -i：提示，交互
#     -f：强制覆盖
```

注意：指定文件名时可以使用文件名通配机制

0x03 文件复制

我们可以使用cp命令来实现文件的复制

```
$cp SRC DEST
$cp 1.txt /tmp    # 将1.txt复制到/tmp目录下
# 假如SRC是一个文件
#      如果目标是一个文件且目标存在：覆盖
#      如果目标文件不存在：创建新文件
#      如果目标存在，且是个目录：复制源至目标目录中，并保持原名

$cp SRC... DEST
$cp ./*.c /tmp   # 将当前目录下的所有.c文件复制到/tmp目录下
# 假如SRC有多个文件：
#     如果目标存在，且是一个文件：复制无法进行
#     如果目标存在，且是一个目录：复制各文件至目标目录中，并保持原名
#     如果目标不存在：复制无法进行
# 如果SRC只有一个且是目录：通常带上-r选项--递归复制，实现cp命令对目录的复制
#     如果目标是一个文件且目标存在：失败
#     如果目标文件不存在：创建新目录
#     如果目标存在，且是个目录：复制源目录到目标目录中，并保持原名

# cp命令的常用选项：
#      -r：递归复制
#      -i：提示，交互
#      -f：强制覆盖
#      -a：-dr 保留所有的文件信息
#      -d：当源为链接文件时，复制链接文件本身，而非指向的源文件
#      -p：保持原有属性
```

注意：指定文件名时可以使用文件名通配机制

0x04 文件查找

参考文档

- Linux find运行机制详解

文件查找的方法有：

- 使用find实时查找：进行实时查找、查找速度慢、进行精确匹配
- 使用locate快速查找：依赖于由自动任务计划每天定时生成的数据库自动进行非实时查找；查找结果精确度较低，但查找速度快，可进行模糊查找；可以通过updatedb命令手动生成数据库

`find`命令查找文件的语法是：`find [options] [查找路径] [查找条件] [处理动作exec]`

- options:

  - `-maxdepth 1`：查找深度为1
- 查找路径：默认为当前目录
- 查找条件：
  - 根据文件名和属主、属组查找
    - 默认查找指定目录下的所有文件
    - `-name "filename"`：查找指定文件名的文件，该文件名格式支持文件名通配机制，严格区分大小写
    - `-iname "filename"`：查找方法同上，但不区分大小写
    - `-user UserName`：根据属主查找,查找属主为UserName的文件
    - `-group GroupName`：根据属组查找，查找属组为GroupName的文件
    - `-uid UID`：根据属主的UID查找
    - `-gid GID`：根据属组的GID查找
    - `-nouser`：查找没有属主的文件
    - `-nogroup`：查找没有属组的文件
  - 根据文件类型查找
    - `-type f`：普通文件
    - `-type d`：目录
    - `-type b`：块设备
    - `-type c`：字符设备
    - `-type l`：符号链接文件
    - `-type p`：命令管道
    - `-type s`：套接字
  - 根据文件大小查找：`-size [+|-]Number[k|M|G]`
    - +表示大于
    - -表示小于
  - 根据文件时间戳查找
    - `-atime [+|-]Number`：根据访问时间来查找，单位为天
    - `-ctime [+|-]Number`：根据改变时间来查找，单位为天
    - `-mtime [+|-]Number`：根据修改时间来查找，单位为天
    - `-amin [+|-]Number`：根据访问时间来查找，单位为分钟
    - `-cmin [+|-]Number`：根据改变时间来查找，单位为分钟
    - `-mmin [+|-]Number`：根据修改时间来查找，单位为分钟
    - 以当前时刻为参照点的过去时间段，2表示`[2,3)`时间段内的操作；+2表示`[3,+oo)`时间段内的操作；-2表示`[0,2)`时间段内的操作
  - 根据文件权限查找：`-perm [+|-]MOD`
    - `MODE`：精确匹配权限
    - `+MODE`：包含或关系，任何一类用户的任何一位权限匹配即可，常用于查找某类用用户的某特定权限是否存在。+444表示至少有一类用户有读权限
    - `-MODE`：包含且关系，每类用户的指定权限位都必须匹配。-444表示每类用户都有读权限

  - 组合条件查找：组合上述查找条件
    - -a：与条件，表示同时满足，默认值可省略；格式为条件1 [-a] 条件2
    - -o：或条件，表示只一个满足，格式为条件1 -o 条件2
    - {-not | !}：非条件，表示条件取反，格式为-not 条件
    - 非的优先级大于与，与的优先级大于或

- 处理动作exec
  - 默认操作是显示查找结果
  - find [查找条件路径] | cpio：将查找结果进行归档
  - find [查找条件路径] | xargs COMMAND：xargs会将管道传递过来的字符串转换为后面命令的传入参数做文件处理；如果没有xargs转换，则通过管道传递过来的字符串只能做文本处理，不能进行文件处理等操作
  - find [查找条件路径] [-print]：默认值，可省略，表示打印到标准输出上。
  - find [查找条件路径] -ls：表示以长格式输出各种文件信息。
  - find [查找条件路径] -exec COMMAND {} \;：把查找到的所有文件一次性地传递给-exec执行指定的命令；{}是用来接收传递值的，同时还可以被后向引用接收到的传递值
  - find [查找条件路径] -ok COMMAND {} \;：把查找到的所有文件一次性地传递给-ok执行指定的命令；{}是用来接收传递值的，同时还可以被后向引用接收到的传递值

find命令的使用用例如下

```
$find ./ -name "*.o" -exec rm {} \;      # 递归当前目录及子目录删除所有.o文件
$find ./ -name '*.o'                     # 查找目标文件夹中是否有obj文件
$find /var/ -user root -a -group mail    # 查找/var目录属主为root且属组为mail的所有文件
$find /usr/ -not \(-user root -o -user bin -o -user hadoop\) # 查找/usr目录下不属
于root、bin或hadoop的所有文件
$find /etc/ -mtime -7 -a -not \(-user root -o -user hadoop\) # 查找/etc/目录下最近一周
内其内容修改过的，且不属于root或hadoop的文件
$find /etc/ -size +1M -a -type f         # 查找/etc/目录下大于1M且类型为普通文件的所有文件
$find /etc/ -not -perm +222              # 查找/etc/目录所有用户都没有写权限的文件
```

locate命令使用方法如下

```
$updatedb               # 更新数据索引库，以获得最新的文件索引信息
$locate STRING          # 根据上述数据索引库查找文件
```

0x05 文件链接

我们可以通过ln命令为文件建立硬链接和软链接

```
$ln /etc/fstab ~/fstab             # 为/etc/fstab文件创建一个硬链接文件
$ln -s /usr/local/docker ~/docker # 为/usr/local/docker目录创建一个软链接文件

# 不能对目录创建硬链接，但是可以创建软链接
# -v选项可以显示链接文件的创建过程
```

0x06 文件压缩

注意：下面压缩工具中“gzip“、bzip2、xz只能压缩文件，不能压缩目录；zip工具既可以用来压缩文件又可以用来压缩目录

gzip压缩工具

```
# gzip工具的压缩文件后缀为.gz
# gzip工具压缩完成后会把原文件给删除，保留压缩文件
$gzip -6 1.txt                      # 指定压缩比1-9，默认的为6，压缩比越大，压缩文件越小
$gzip -c 1.txt >> foo.gz            # -c选项将压缩内容重定向至指定的压缩文件，这样可以实现压缩后
保留源文件
$gzip -c 1.txt 2.txt > foo.gz       # 同时压缩多个文件输出到指定压缩文件
$cat 1.txt 2.txt | gzip > foo.gz    # 实现功能和上面一样，但是压缩效果要好
$zcat foo.gz                        #查看.gz压缩文件中的内容,它会创建一个临时目录将压缩文件解压
并提供访问查看，类似于cat查看普通文件内容一样
```

bzip2压缩工具

```
# bzip2工具的压缩文件后缀为.bz2
# 对于大文件的压缩，bzip2的压缩效果比gzip要好
# bzip2工具压缩完成后会把原文件给删除，保留压缩文件
$bzip2 -k 1.txt                       # 压缩后保留原文件，不会删除
$bzip2 -6 1.txt                       # 指定压缩比1-9，默认的为6，压缩比越大，压缩文件越小
$bzip2 -c 1.txt >> foo.bz2            # -c选项将压缩内容重定向至指定的压缩文件，这样可以实现压
缩后保留源文件
$bzip2 -c 1.txt 2.txt > foo.bz2       # 同时压缩多个文件输出到指定压缩文件
$cat 1.txt 2.txt | bzip2 > foo.bz2    # 实现功能和上面一样，但是压缩效果要好
$bzcat foo.bz2                        #查看.bz2压缩文件中的内容,它会创建一个临时目录将压缩文件
解压并提供访问查看，类似于cat查看普通文件内容一样
```

xz压缩工具

```
# xz工具的压缩文件后缀为.xz
# 该工具的压缩效果是最好的
# xz工具压缩完成后会把原文件给删除，保留压缩文件
$xz -k 1.txt                        # 压缩后保留原文件，不会删除
$xz -6 1.txt                        # 指定压缩比1-9，默认的为6，压缩比越大，压缩文件越小
$xz -c 1.txt >> foo.xz              # -c选项将压缩内容重定向至指定的压缩文件，这样可以实现压缩后保
留源文件
$xz -c 1.txt 2.txt > foo.xz         # 同时压缩多个文件输出到指定压缩文件
$cat 1.txt 2.txt | xz > foo.xz      # 实现功能和上面一样，但是压缩效果要好
$xzcat foo.xz                       #查看.xz压缩文件中的内容,它会创建一个临时目录将压缩文件解
压并提供访问查看，类似于cat查看普通文件内容一样
```

zip压缩工具：在这里只介绍如何压缩文件

```
# 该工具的压缩文件后缀为.zip
# 该工具属于归档压缩工具(先对文件进行归档然后进行压缩)，所有不会在压缩或者解压时删除原文件
$zip message.zip message          # 将message文件压缩成message.zip文件。需要指定压缩文件名
$zip -6 message.zip message       # 指定压缩比1-9，默认的为6，压缩比越大，压缩文件越小
$find . -name "*.[ch]" -print | zip source -@   # 将当前目录下所有.c和.h源文件归档压缩；-
↪@选项使zip不再从命令行中获取待压缩文件路径，而是从标准输入或管道中获取
```

0x07 文件解压

此处的文件解压工具与上述的压缩工具一一对应

gunzip是gzip压缩工具的解压工具，只解压.gz压缩文件

```
# 解压缩后会把压缩文件删除，保留原文件
$gunzip 1.txt.gz
$gunzip -c 1.txt.gz > 2.txt           # -c选项将解压内容重定向至指定文件，这样可以实现解压后保
留源压缩文件
$gzip -d 1.txt.gz                     # 类似于gunzip
$gzip -cd old.gz | gzip > new.gz      # 解压压缩文件然后压缩到另一压缩文件中，源压缩文件不变
$gzip -d old.gz | gzip > new.gz       # 解压压缩文件然后压缩到另一压缩文件中，源压缩文件会删
除，保留解压后原文件
```

bunzip2是bzip2压缩工具的解压工具，只解压.bz2压缩文件

```
# 解压缩后会把压缩文件删除，保留原文件
$bunzip2 1.txt.bz2
$bunzip2 -k 1.txt.bz2                    # 解压后不会删除压缩文件
$bunzip2 -c 1.txt.bz2 > 2.txt            # -c选项将解压内容重定向至指定文件，这样可以实现解压后
保留源压缩文件
$bzip2 -d 1.txt.bz2                      # 类似于bunzip2
$bzip2 -cd old.bz2 | gzip > new.bz2      # 解压压缩文件然后压缩到另一压缩文件中，源压缩文件不变
$bzip2 -d old.bz2 | gzip > new.bz2       # 解压压缩文件然后压缩到另一压缩文件中，源压缩文件会删
除，保留解压后原文件
```

unxz是xz压缩工具的解压工具，只解压.xz压缩文件

```
# 解压缩后会把压缩文件删除，保留原文件
$unxz 1.txt.xz
$unxz -k 1.txt.xz                    # 解压后不会删除压缩文件
$unxz -c 1.txt.xz > 2.txt            # -c选项将解压内容重定向至指定文件，这样可以实现解压后保留
源压缩文件
$xz -d 1.txt.xz                      # 类似于unxz
$xz -cd old.xz | gzip > new.xz       # 解压压缩文件然后压缩到另一压缩文件中，源压缩文件不变
$xz -d old.xz | gzip > new.xz        # 解压压缩文件然后压缩到另一压缩文件中，源压缩文件会删除，
保留解压后原文件
```

unzip是zip压缩工具的解压工具，只解压.zip压缩文件

```
# 该工具属于归档压缩工具(先对文件进行归档然后进行压缩)，所有不会在压缩或者解压时删除原文件
$unzip message.zip           # 解压缩到当前目录下
$unzip message.zip -d test/ # 解压缩到test目录下
```

目录文件

命令汇总

- *mkdir*
- *rm*
- *mv*
- *cp*
- *find*
- *locate*
- *ln*
- *cd*
- *pwd*
- *tar*
- *zip*
- *unzip*
- *ls*

目录文件的基本操作有

- 目录创建
- 目录删除
- 目录移动

- 目录复制
- 目录查找
- 目录链接
- 目录切换
- 目录归档
- 目录压缩
- 目录解压
- 目录查看

0x00 目录创建

我们通常使用mkdir命令来创建目录

```
$mkdir test                  # 如果目录存在将会提示错误
$mkdir -pv test/docker       # -p表示如果父目录不存在则先创建父目录再创建子目录，父目录存在也不会提示错误；-v表示显示创建的过程
$mkdir -m=777 test           # 指定创建目录的权限模型为777

# 创建级联目录
$mkdir -pv baklog/{bin/{test,docker},lib,log/{cep,dod,testlog}}   # 同一级目录名放在同一个大括号里，目录名用逗号分隔，但是之前不能留有空白
```

0x01 目录删除

目录删除使用rm命令

```
$rm -rf test              # 强制删除test目录以及其子目录或文件
```

0x02 目录移动

目录移动使用move命令，可参考文件移动mv

```
$mv baklog/ test          # 将baklog目录以及子目录或文件移到到test目录下，此处可以添加-r选项，可以不添加，默认是递归移动
```

0x03 目录复制

目录移动使用cp命令，可参考文件移动cp

```
$cp -r baklog/ bak      # 将baklog目录下子目录或者文件复制到bak目录下，此处需要使用-r选项才能实现递归复制
```

0x04 目录查找

目录查找使用find命令和locate命令，可参考文件查找find和文件查找locate

0x05 目录链接

目录只能创建软链接，不能创建硬链接，同样使用ln命令，可参考文件链接ln

0x06 目录切换

我们可以通过cd命令实现目录的切换

```
$cd test              # 切换到test目录下
$cd -                 # 切换到上一工作目录
$cd ~                 # 切换到家目录
$cd ..                # 切换到上级父目录
$cd ../../            # 切换到父目录的父目录
$pwd                  # 显示当前路径
```

0x07 目录归档

我们知道gzip、bzip2、xz三个压缩工具只能对文件进行压缩，不能对目录进行压缩；此时我们可以通过tar工具将目录进行归档打包成一个文件，便于这三个压缩工具对目录进行压缩

tar命令的使用语法为：`tar [OPTION...] [FILE]...`，其中常用的OPTIONS有

- `-c`：创建归档
- `-f`：指定归档后的文件名。归档后的文件名必须跟在该选项后面
- `-x`：展开归档
- `-v`：显示命令执行的过程
- `-t`：不展开而直接查看被归档的文件
- `-z`：使用gzip压缩
- `-j`：使用bzip2压缩
- `-J`：使用xz压缩
- `-C`：指定解压缩后文件的存放路径

常见用法有

```
# 该工具属于归档压缩工具(先对文件进行归档然后进行压缩)，所有不会在压缩或者解压时删除原文件

$tar -cf archive.tar foo bar      # 将foo和bar进行归档成archive.tar文件(foo和bar可以是目
录或者文件)
$tar -tvf archive.tar             # 不展开archive归档文件直接列出其中的文件列表
$tar -xf archive.tar              # 展开archive归档文件
$tar -zcf archive.tar.gz FILE     # 将FILE目录或文件先进行归档，然后使用gzip将其压缩成.gz压
缩文件
$tar -jcf archive.tar.bz2 FILE    # 将FILE目录或文件先进行归档，然后使用bzip2将其压缩成.bz压
缩文件
$tar -Jcf archive.tar.xz FILE     # 将FILE目录或文件先进行归档，然后使用xz将其压缩成.xz压缩
文件
$tar -zxf archive.tar.gz          # 将指定的.gz压缩文件先使用gzip -d解压缩，然后将归档文件展
开成对应的原文件或目录
$tar -jxf archive.tar.bz2         # 将指定的.bz2压缩文件先使用bzip2 -d解压缩，然后将归档文件
展开成对应的原文件或目录
$tar -Jxf archive.tar.xz          # 将指定的.xz压缩文件先使用xz -d解压缩，然后将归档文件展开
成对应的原文件或目录
```

0x08 目录压缩

此处目录压缩有两种方法

- 使用zip压缩工具

- 使用tar归档工具

```
# 使用tar压缩目录
# 该工具属于归档压缩工具(先对文件进行归档然后进行压缩)，所有不会在压缩或者解压时删除原文件
$tar -zcf archive.tar.gz FILE     # 将FILE目录或文件先进行归档，然后使用gzip将其压缩成.gz压
缩文件
$tar -jcf archive.tar.bz2 FILE    # 将FILE目录或文件先进行归档，然后使用bzip2将其压缩成.bz压
缩文件
$tar -Jcf archive.tar.xz FILE     # 将FILE目录或文件先进行归档，然后使用xz将其压缩成.xz压缩
文件

# 使用zip压缩目录
# 该工具属于归档压缩工具(先对文件进行归档然后进行压缩)，所有不会在压缩或者解压时删除原文件
# 当.zip文件存在时，归档压缩模式类似于文件打开的追加模式，存在的文件则会覆盖，不存在的则会新建添加
$zip pam.d.zip pam.d/*            # 将pam.d目录下所有目录或文件先归档然后压缩成pam.d.zip文
件，不包括pam.d目录名，同时不会进行目录递归
$zip pam.d.zip pam.d/             # 只将pam.d目录名进行归档然后压缩成pam.d.zip文件，不包括
其下的子目录和文件
$zip -r pam.d.zip pam.d/          # 将pam.d目录名以及其所有子目录或者文件先归档然后压缩
成pam.d.zip文件，此时会进行目录递归
```

0x09 目录解压

tar工具只能解压.gz、.bz2、.xz格式的压缩文件

```
# 该工具属于归档压缩工具(先对文件进行归档然后进行压缩)，所有不会在压缩或者解压时删除原文件
$tar -zxf archive.tar.gz          # 将指定的.gz压缩文件先使用gzip -d解压缩，然后将归档文件展
开成对应的原文件或目录
$tar -jxf archive.tar.bz2         # 将指定的.bz2压缩文件先使用bzip2 -d解压缩，然后将归档文件
展开成对应的原文件或目录
$tar -Jxf archive.tar.xz          # 将指定的.xz压缩文件先使用xz -d解压缩，然后将归档文件展开
成对应的原文件或目录
$tar -zxf archive.tar.gz -C tet/ # 解压缩到tet目录下
```

.zip压缩文件只能使用unzip工具实现解压

```
# 该工具属于归档压缩工具(先对文件进行归档然后进行压缩)，所有不会在压缩或者解压时删除原文件
$unzip message.zip           # 解压缩到当前目录下
$unzip message.zip -d test/ # 解压缩到test目录下
```

0x10 目录查看

我们可以通过ls命令来查看目录中的内容

```
$ls                 # 列出当前目录下的内容(不包括隐藏文件)
$ls -a              # 列出当前目录下所有有文件，包含隐藏文件
$ls -A              # 列出当前目录下所有文件，但不显示.(当前目录)和..(上级目录)通用路径
$ls --color[=never|auto|always]   # 显示颜色
$ls -l              # 长格式显示
$ls -d              # 只显示当前目录(.)自身的属性，通常与-l同时使用
$ls -r              # 实现逆序显示
$ls -R              # 实现递归显示，效果跟tree命令很相似，显示子目录中的子目录或者文件
$ls -i              # 显示目录下各文件对应的存储在元数据区的inode索引号
$ls -h              # 文件大小以人类易读格式显示
```

文本处理

以下所有的操作对象都是：**文本文件**

目录

文本查看

命令汇总

0x00 cat

cat命令会读取文本所有内容输出到标准输出上

```
$cat -E 1.txt              # 输出文本内容时，显示行结束符$
$cat --show-ends 1.txt     # 效果同上
$cat -v 1.txt              # 显示非打印符
$cat -e 1.txt              # 效果等同于-vE
$cat -n 1.txt              # 显示每行按顺序编号
$cat -s 1.txt              # 将多个连续的空白行合并显示一个空白行
```

0x01 tac

tac的功能和cat相似，只不过是按行逆序显示文件

```
$tac              # 当没有指定文件时，从标准输入读取内容，按两次ctrl+d提交内容
$tac 1.txt        # 逆序显示指定文件内容
```

0x02 head

head命令会读取文件的首部内容输出到标准输出上

```
$head 1.txt          # 显示1.txt的前10行
$head -n 20 1.txt    # 显示1.txt的前20行
$head -c 30 1.txt    # 显示1.txt的前30个字节
```

0x03 tail

tail命令会读取文件的尾部内容输出到标准输出上

```
$tail 1.txt          # 显示1.txt的尾部10行
$tail -n 20 1.txt    # 显示1.txt的尾部20行
$tail -c 100 1.txt   # 显示1.txt的尾部100个字节
$tail -f 1.txt       # 当文件内容增加时，同步输出到标准输出上；常用来监控日志
```

0x04 more

more命令支持读取文件的全部内容，然后分屏显示；但是该命令只支持向后翻屏，当翻到文件尾部时不支持向前翻

```
$more 1.txt       # 分屏显示1.txt文件内容
$more +10 1.txt   # 从文件的第10行开始分屏显示
```

使用more命令分屏查看时可以用到的快捷键有：

- h、?：查看可以使用的快捷键
- q：退出more文档
- space：向后翻一屏
- enter：向后翻一行
- v：用vim打开当前文件
- =：打印当前行数
- :f：打印当前文件名和行数
- .：重新执行上次快捷键

0x05 less

less命令类似于more命令，读取文件全部内容，然后分屏显示；不同的是，less命令支持前后翻屏。我们通常查看命令帮助使用的man文档就是使用less命令打开的

```
$less 1.txt         # 分屏显示1.txt内容
```

使用less命令分屏查看时可以用到的快捷键有：

- h、H：查看可以使用的快捷键
- q、:q：退出less文档
- enter：向后翻一行
- y：向前翻一行
- space：向后翻一屏
- b：向前翻一屏
- d: 向后翻半屏
- u：向前翻半屏
- /pattern：向后模式匹配
- ?pattern：向前模式匹配
- &pattern：只显示被模式匹配到的行
- n：向模式匹配的方向查看匹配内容
- N：向模式匹配相反的方向查看匹配内容
- g：跳转到文件首行

- G：跳转到文件末行

0x06 wc

wc的全称为word count，该命令用来做字符数统计的

```
$wc                 # 等待标准输入内容，按两次ctrl+d提交，显示行数，单词书和字节数
$wc -l 1.txt        # 统计行数
$wc -w 1.txt        # 统计单词数
$wc -c 1.txt        # 统计字符数
```

0x07 tr

tr命令是用来转换或删除字符；但是该命令不能接受文件作为参数，只能接受字符作为参数，所以必须使用命令管道进行参数传递，因为管道传递过来的是字符流。其语法格式：tr [options] SET1 [SET2]

- options常用选项
  - 不使用任何选项且SET1和SET2都存在则表示将SET1的内容转换成SET2中对应的内容；转换规则是：SET1中第n个字符转换成SET2中对应的第n个字符，如果SET2的长度小于SET1，则重复SET2中的最后一个字符来补足其长度
  - -d：删除SET1匹配到的内容，此时没有SET2
  - -s：压缩SET1匹配到的内容，此时没有SET2
- SET1/SET2都是字符串，它们支持的形式有
  - \\
  - \b：删除符
  - \n：换行符
  - \r：回车符
  - \t：制表符
  - \v：垂直制表符
  - CHAR1-CHAR2：例如0-9就是从0到9
  - [:alnum:]：所有字母和数字
  - [:alpha:]：所有字母
  - [:blank:]：所有水平留白
  - [:cntrl:]：ctrl字符
  - [:digit:]：所有数字
  - [:graph:]：所有可打印字符，不包括空格
  - [:lower:]：所有小写字母
  - [:print:]：所有可打印字符，包括空格
  - [:punct:]：所有标点符号
  - [:space:]：所有垂直留白和水平留白
  - [:upper:]：所有大写字母
  - [:xdigit:]：所有十六进制数

```
# 字符转换
$echo 12345 | tr '0-9' '9876543210'          # 将12345转换成87654，输出到标准输出上
$cat 1.txt | tr '\t' ' '                     # 将1.txt文件中的制表符转换成空格，输出到标准输
出上
$cat /etc/issue | tr 'a-z' 'A-Z'             # 将/etc/issue中的所有小写字母转换成对应的大写
字母，输出到标准输出上

# 字符删除
$cat 1.txt | tr -d '0-9'                     # 删除1.txt文件中所有数字，输出到标准输出上

# 字符压缩
$cat 1.txt | tr -s 'a'                       # 将1.txt文件中的连续a(例如：aaaaa)压缩成一
个a，输出到标准输出上
```

0x08 sort

sort命令将会对文件行内容进行排序，然后输出到标准输出上

常用的选项有：

- -f：排序时忽略大小写
- -b：排序时忽略前导空白符
- -n：按照指定数字段的大小对文本文件进行行升序排序
- -t：后面紧跟字符，指定行分隔符
- -k：后面紧跟数字，指定分隔后进行比较字段
- -u：重复的行，只显示一行
- -d：按字典序进行排序
- -r: 逆序排序

```
$sort /etc/passwd              # 默认通过比较行首字符在ASCII表中的次序对文本文件进行行升序排序
$sort -n -t: -k3 /etc/passwd   # 以:作为分隔符分隔每行内容，按照第3段(数字段)的大小进行升序排
序
$sort -rbd 1.txt                # 忽略像空格之类的前导空白字符，按照字典进行降序排序
```

0x09 uniq

uniq命令用来移除重复的行(只有连续的相同行才是重复行，不连续的相同行不是重复行)，所以通常和sort命令配合使用，因为只有排序后才能保证相同的行是连续的

```
$sort 1.txt | uniq              # 消除重复的行
$sort 1.txt | uniq -s 10 -w 10 # 消除重复的行，从第10(-s)个字符开始往后比较10(-w)个字符，如
果相同则为重复的行
$sort 1.txt | uniq -c           # 统计每一行出现的次数
$sort 1.txt | uniq -d           # 仅显示出现次数至少两次的行
$sort 1.txt | uniq -u           # 仅显示不重复的行
```

文本查找

命令汇总

- *cut*
- *grep*

这些工具可以称之为文本过滤器，它们会根据用户指定的文本模式对目标文件或目标字节流进行逐行搜索匹配，并输出匹配文本模式的整行信息或特定字段

0x01 cut

`cut`命令会根据指定的分隔符对每行文本内容进行切片，并显示出每行需要的切片

命令常用选项有：

- `-d`：后面紧跟字符，将该字符指定为行分隔符；默认是以`tab`键为分隔符
- `-f`：后面紧跟数字，以字段为单位显示文本每行内容；一般和`-d`选项指定分隔符一起使用
- `-b`：后面紧跟数字，以字节为单位显示文本每行内容，不需要指定分隔符
- `-c`：后面紧跟数字，以字符为单位显示文本每行内容，不需要指定分隔符
- `--complement`：显示文本每行没有被`-b`、`-c`、`-f`选项匹配到的内容
- `-s`：不显示没有被`-f`选项匹配到的行，和`-f`选项一起使用

`-b`、`-c`、`-f`选项后面数字都支持以下形式：

- 一个字段、字节或字符使用一个数字即可
- 多个离散字段、字节或字符使用`,`分隔多个数字
- 多个连续字段、字节或字符使用`-`连接两个数字
  - `N-`表示第N个字段、字节或字符到结尾
  - `-M`表示第1到第M个字段、字节或字符
  - `N-M`表示第N到第M个字段、字节或字符

```
$cut -f2 1.txt                  # 默认以tab为分隔符
$cut -f2 -s 1.txt               # 如果没有第2字段，则不打印该行内容
$cut -d: -f1 /etc/passwd        # 以:分隔/etc/passwd每行内容，打印每行分隔后的第一个字段
$cut -c1-5 /etc/passwd          # 打印每行第1到第5个字符
$cut -b-2 /etc/passwd           # 打印每行前2个字节内容
$cut -c1 --complement 1.txt     # 打印每行除第1个字符之外的所有字符
```

0x01 grep

参考文档

- grep命令中文手册

man文档关于grep命令的介绍是：global search basic regular expression（RE）and print out the line，即基本正则表达式搜索，输出整行信息。它可接收文件或字节流作为搜索对象

grep命令使用的语法格式：grep [OPTIONS] PATTERN [FILE...]

- OPTIONS常用选项
  - --color[=WHEN]：将匹配到文本模式的信息以指定的颜色显示出来。默认auto值是红色
  - -v：反向匹配，显示不能被模式匹配到的行
  - -o：仅显示被模式匹配到的字串，而非整行
  - -c：统计文件中包含文本的次数
  - -n：打印文本匹配的行号以及文件名
  - -I：只打印文件名
  - -r：多级目录中对文本递归搜索，不包括符号链接文件
  - -R：多级目录中对文本递归搜索，包括符号链接文件
  - -i：不区分大小写，ignore-case
  - -E：默认只支持基本正则表达式，如果想使用扩展正则表达式需要用-E选项
  - -A num：不仅显示匹配到的行，还显示模式下面的num行
  - -B num：不仅显示匹配到的行，还显示模式上面的num行
  - -C num：不仅显示匹配到的行，前后各显示num行
- PATTERN：查找匹配模式
  - 基本正则表达式BRE使用-e选项，但是由于基本正则是默认支持的，所有可以省略
  - 如果想要支持扩展正则表达式ERE，则需要在PATTERN之前加上-E选项
  - 如果想要支持Perl正则表达式PCRE，则需要在PATTERN之前加上-P选项
- FILE：查找匹配对象
  - 可以是文件名
  - 也可以是管道送过来的字节流

注意：PATTERN中一旦使用正则表达式的元字符(^,$等)作为文本模式，就需要用引号括起来：

- 单引号，表示强引用，不允许做变量替换
- 双引号，表示弱引用，允许做变量替换

```
$grep -R -n "class" .                  # 多级目录中对class文本进行递归搜索，打印所在文件，行号以及整行内容
$grep -e "class" -e "vitural" file   # 匹配多个模式
$grep -rnP "\xE4\xB8\xAD\xE6\x96\x87|\xD6\xD0\xCE\xC4" . # 使用perl正则在多级目录中递归搜索utf-8编码和gb2312编码分别是E4B8ADE69687和D6D0CEC4的中文
$cat LOG.* | tr a-z A-Z | grep "FROM" | grep "WHERE" > b # 将日志中的所有带where条件的sql查找查找出来
```

0x02 egrep

egrep命令的使用方法和grep命令完全一样，唯一区别就是：egrep命令使用扩展正则表达式作为默认文本模式，相当于grep -E

0x03 awk

参考文档

- awk知识点全回顾

awk是一个报告生成工具(过滤显示工具)，它的工作机制是：使用指定的分隔符将读取的每一行数据进行切割，然后根据PATTERN文本模式匹配每一行内容，最后执行action动作处理匹配到的行内容

awk的分支有

- awk：早期使用
- nawk(new awk)：付费使用
- gawk(GNU awk)：linux平台使用，在linux上awk只是gawk的一个符号链接

`awk`命令的语法格式有两种：

- `awk [OPTIONS] -f program-file [ -- ] FILE ...`
- `awk [OPTIONS] [ -- ] program FILE ...`

两种语法大同小异，`program-file`只是文件内容为`program`的文件名而已。其中`FILE`是`awk`命令的操作对象，可以是以空格分隔的多个文件名，也可以是管道传送过来的字符流，甚至可以是赋值变量等；至于`OPTIONS`和`program`需要详细说明下

0x0300 OPTIONS

常用选项有：

- `-F`：指定输入行的字段分隔符，也可以通过设置内置变量`FS`实现，默认字段分隔符为空白符，也可通过正则表达式指定分隔符
  - `-F " "`：默认的，会压缩所有前导空白，包括制表符和空格
  - `-F " :"`：当空格后跟一个冒号时作为分隔符。会压缩前导空格，但不会匹配制表符，更不会压缩制表符
  - `-F "[ ]"`：只表示一个空格，不压缩任何空白
  - `-F "|"`：指定竖线作为分隔符
  - `-F ",[ \t]*|[ \t]+"`：逗号后跟0或多个空白，或者只有1或多个空白时作为分隔符

0x0301 program

`program`是`awk`的重中之重，称为awk的程序，它的格式为：`'BEGIN{ACTIONS}PATTERN{ACTIONS}END{ACTIONS}`注意此处有单引号，由此得出awk详细的执行流程是：

- `awk`读取`program`后面第一个文件第一行之前执行`BEGIN`后面的`ACTIONS`程序，该程序通常用于输出一个标题，或者初始化一些格式、变量等
- `awk`每读取`program`后面文件的一行内容就使用`BEGIN`或`OPTIONS`中定义的输入字段分隔符和输入行分隔符对行内容进行段分隔和行分隔，然后将该行内容与`PATTERN`文本模式进行匹配比较，如果行内容能够匹配上则执行`PATTERN`后面的`ACTIONS`程序

- awk处理完program后面最后一个文件的最后一行后执行END后面的ACTIONS程序，该程序通常用于最后的总结性输出

在program中

- BEGIN{ACTIONS}字段不用提供输入流，BEGIN是固定字样，类似于类中的构造函数入口，ACTIONS是构造函数的函数体，其语法和PATTERN对应的ACTIONS语法一致，功能主要是初始化
- PATTERN{ACTIONS}是核心字段，称之为主输入循环(main input loop)，在进入主输入循环之前，可以不用提供输入流，但进入主输入循环后，必须提供输入流。
  - PATTERN称之为文本模式，类似于类中特定函数方法的调用入口，只有满足该文本模式的内容，才能调用其对应的执行函数体
  - ACTIONS称之为执行动作，类似于上述函数方法的函数体，只有满足前面的文本模式的内容，才能调用该函数体
  - 其中PATTERN或ACTIONS二者可省一：省略PATTERN时表示对所有输入流都执行ACTIONS，省略ACTIONS表示对符合条件的输入流都执行默认的print动作
- END{ACTIONS}字段不用提供输入流，END是固定字样，类似于类中的析构函数入口，ACTIONS是析构函数的函数体，其语法和PATTERN对应的ACTIONS语法一致，功能主要是收尾处理

由上述可知，我们需要关注的只有PATTERN文本模式以及BEGIN/PATTERN/END对应的ACTIONS执行动作

PATTERN模式

也称为文本模式，用来过滤输入流，只有匹配文本模式的输入流才能执行PATTERN对应的ACTIONS工作，文本模式的形式有以下几种

- /regexp/：正则匹配模式，regexp为正则表达式，需要使用//将其括起来，有两种匹配表达式：
  - /regexp/：表示当前行内容能被regexp匹配则为真
  - !/regexp/：表示当前行内容不能被regexp匹配就为真
  - 例如：`awk -F : '/^root\>/{print $1,$7}' /etc/passwd`打印/etc/passwd文件中以root开头的行
- expression：表达式匹配模式，expression是由操作符合左右数组合而成，expression不需要加斜线，且expression中操作符、左右数之间没有空格
  - 支持的操作符有：
    - 比较操作符有：
      - 数值比较：<、<=、==、!=、>=、>
      - 正则匹配：~表示能被右数/regexp/匹配模式匹配，如$7~/bash$/
      - 正则反匹配：!~表示不能被右数/regexp/匹配模式匹配，如$7!~/bash$/
    - 算术操作符有：+、-、*、/、%、^(取幂)、**(取幂，非POSIX标准，不可移植)
    - 逻辑操作符有：&&、||、!，如`$4 == "Asia" && $3 > 500, ! (NR > 1 && NF > 3)`
  - 操作符左数可以是
    - 使用字段变量：$0表示整行、$1~$n分别表示被输入字段分隔符分隔号的第1字段和第n字段
    - 使用内置变量
  - 操作符右数可以是

        - 如果左数是数值，则操作符一般是数值比较，右数一般也是数值
        - 如果左数是字符串，则操作符一般是正则匹配，右数一般是/regexp/
    - 例如：`awk -F : '$7~/bash$/{print $1}' /etc/passwd`打印/etc/passwd文件中第7段是以bash结尾的行的第1段
- 地址定界模式，该模式有两种形式
    - `/regexp1/,/regexp2/`：第一次被`regexp1`匹配到的行开始到第一次被`regexp2`匹配到的行结束，这些内容都可以执行`PATTERN`后面的`ACTIONS`动作
    - `expression1,expression2`：第一次满足`expression1`的行开始到第一次满足`expression2`的行结束，这些内容都可以执行`PATTERN`后面的`ACTIONS`动作

需要注意的是：`regexp`正则表达式可以被赋值给一个变量，然后引用该变量来匹配数据

```
reg="^[0-9]+$"
$2~reg
```

`awk`不需声明变量数据类型，它内置字符串类型和数值类型

ACTIONS支持的变量有以下几类

- 普通变量：也可以称为自定义变量
    - 如果要赋值字符串给自定义变量，则应该使用双引号将其括起来：`reg="^[0-9]+$"`
        - `name = "abc" "bcd"`等价于`name="abcbcd"`,可以将空格理解为awk的拼接字符，因为awk会忽略任何不被引号包围的空白
    - 如果要赋值数值给自定义变量，则不需要使用双引号：`reg=9`
- 字段变量：每行内容被输入字段分隔符分隔形成的变量
    - `$0`：表示整行内容
    - `$1~$n`：表示每行使用分隔符分隔后的第一字段~第n字典
- 内置变量：该变量可分为两类
    - awk内部自动修改的变量
        - `ARGV`：命令行参数数组，从0开始计数直到ARGC-1
        - `ARGC`：ARGV数组元素的个数
        - `FILENAME`：当前处理的文件名
        - `FNR`：当前处理文件的记录号(行号)(file record num)
        - `NR`：已处理的总记录数(总行数)，多个文件时不重置(record num)
        - `NF`：当前行使用分隔符分隔完后的字段总数(field num)
    - awk内部不会改动的系统变量，完全需要手动修改，这类一般都有默认值
        - `FS`：输入字段分隔符，默认为空白(field separate)
        - `OFS`：输出字段分隔符，默认为空白(output field separate)
        - `RS`：输入流记录(行)分隔符，默认为`\n`，该变量只取变量值的第一个字符(record separate)，若设置为`\t\t`，则第二个`\t`被忽略
        - `ORS`：输出流记录(行)分隔符，默认为`\n`，该变量可识别多字符(output record separate)
        - `OFMT`：printf输出数值转换成字符串输出时的格式，默认为`%.6g`
        - `CONVFMT`：printf输出数值转换成字符串输出时的格式，会被OFMT覆盖，默认为`%.6g`
        - `RLENGTH`：被match函数匹配的字符串的长度

        * RSTART：被match函数匹配的字符串的开始位置
        * SUBSEP：下标分隔符，默认为\034，ASCII中034代表的是双引号"

- 数组变量：awk数组和shell数组类似，都支持数值index的普通数组和字符串index的关联数组，其实数值index仍然会转换成字符串index，所以awk的数组类型都是关联数组
  - 数组格式：array_name[index]
  - 数组访问
    * 获取数组元素
      · array_name["var"]：index为var的数组元素，若该数组元素没有定义，则会定义一个新的数组元素array_name[""]
      · array_name[var]：index为变量var的值的数组元素，若该数组元素没有定义，则会定义一个新的数组元素array_name[""]
    * 判断数组元素是否存在
      · if ("var" in array_name)：判断数组array_name中是否有var下标对应的数组元素。如果有，它会返回1，否则返回0
      · if (array_name["var"] != "")：判断数组变量的值是否为空也可判断该数组元素是否存在，但当该元素不存在时，会创建它，一般不采用这种方式来判定
    * 循环遍历数组：for (i in array_name){do something about array_name[i]}
      · 变量i用来遍历数组的index，array_name是数组名
      · 这种方法是以遍历index的方式来遍历数组。由于index的顺序随机，所以遍历时顺序也是随机的
    * 删除数组元素或数组
      · delete array_name["var"]：删除array_name中下标为var的元素
      · delete array_name：删除数组array_name

ACTIONS接收变量的途径有

此处所说的变量主要是指外界变量，例如：shell中的变量、shell中命令执行的结果、开始执行awk前应该初始化的变量等

1. **将待传递变量当作文件名被awk解析**

变量赋值语句定义位置：program之后

该种定义方式的特点是：变量不可在BEGIN中使用，因为它是被当做文件解析的，只有在需要读取主输入文件的时候才会被解析，也就是说执行完BEGIN后才会来解析赋值变量

```
awk 'BEGIN{}PATTERN{print var1,var2,var3}' var1=value1 var2=value2 file1␣
↪var3=value3 var1=value4 file2
```

在上面的语句中

- 当awk执行完BEGIN程序后，准备读取主输入，于是开始解析program后的输入文件
- 解析时发现，var1和var2都是赋值语句，于是当成变量处理
- 当读取到file1时，发现只有一个参数，则当作输入文件，于是开始处理该文件。在处理file1时，var1和var2都是有效的，但var3还未赋值，因此var3无效
- 当处理完file1后，继续解析下一个主输入文件，此时var3被赋值，并开始处理file2
- 在处理file2时，var1、var2和var3都是有效的，但var1被新值覆盖

此外，还可以将shell命令的结果赋值给这些预定义变量。如下展示了几种变量定义的方式：

```
name="Ma longshuai"
awk 'program' OFS=":" var1="$name" var2="`echo Ma longshuai2`" var3="Ma longshuai3
↪" var4=Malongshuai4 filename
```

2. 使用-v选项传递变量

变量赋值语句定义位置：program之前

该种定义方式的特点是：变量可以在BEGIN中使用，因为它定义在program之前，先解析赋值变量，然后执行BEGIN

每定义一个变量，都需要使用一个-v选项，例如：

```
name="Ma longshuai"
awk -v OFS=":" -v var1="$name" -v var2="`echo Ma longshuai2`" -v var3="Ma␣
↪longshuai3" 'program' filename
```

3. 通过参数数组ARGV传递变量

ARGV是内置的数组变量。awk内部会将**命令行**切分，并按规则将各参数存放到ARGV数组中，数组下标从0开始，这是awk中唯一下标从0开始的数组。在存放到“ARGV“时，所有的选项和program会被忽略。

每存储一个数组变量，特殊变量ARGC的值增加1。因此ARGC的值代表的是参数的个数。所以，数组变量从ARGV[0]到ARGV[ARGC-1]

可使用类似下面的循环来遍历ARGV数组

```
awk -F "\t" -v var1="value1" 'BEGIN{
        for(i=0;i<ARGC;++i){
                print "ARGV[" i "]: " ARGV[i]
        }
        print "ARGC: " ARGC
}' "a" "b" "v=1" file

# 输出结果
ARGV[0]: awk
ARGV[1]: a
ARGV[2]: b
ARGV[3]: v=1
ARGV[4]: file
ARGC: 5
```

注意，ARGV[0]存储的是awk命令，-F和-v选项都没有存储到ARGV中

ARGC和ARGV数组变量的值都可以手动修改。命令行分割存储完成之后，开始处理BEGIN，再处理主循环输入。因此，在BEGIN里修改ARGV中输入文件对应的值，可以改变awk所读取的输入文件，若将其设置为空，则该数组变量直接被跳过，也就不再读取该输入文件

需要注意的是，当增加ARGV元素时，必须同时递增ARGC的值，因为awk是根据AGRC来读取ARGV的。同理，只增加ARGC的值，将导致新建ARGV数组元素，且这些新元素的值为空。也因此，如果减小ARGC的值，将导致无法访问超出ARGC-1边界的ARGV元素。

awk的ACTIONS中支持的运算符有

- 比较操作符：
  - 数值比较：<、<=、==、!=、>=、>
  - 正则匹配：~表示能被右数/regexp/匹配模式匹配，如$7~/bash$/
  - 正则反匹配：!~表示不能被右数/regexp/匹配模式匹配，如$7!~/bash$/
- 算术操作符：+、-、*、/、%、^(取幂)、**(取幂，非POSIX标准，不可移植)

- 赋值操作符：++、--、+=、-=、*=、/=、%=、^=、**=awk支持复合赋值，例如FS = OFS = "\t"表示输入字段分隔符和输出字段分隔符都被赋值为制表符
- 逻辑操作符：&&、||、!，如$4 == "Asia" && $3 > 500，! (NR > 1 && NF > 3)

awk的ACTIONS中支持的控制流语句有

- 赋值语句
- 条件判断语句
- 循环语句

此处的赋值语句主要指的是对变量的赋值操作，不同赋值语句之间使用;隔开

- FS="\n";RS=""
- name="test anony"

条件判断语句的格式有

```
# if多行格式
if(测试表达式){
        cmd1
        cmd2
        ...
}

# if单行格式
if(测试表达式){cmd1;cmd2;...}

# if-else单行格式
if(测试表达式){cmd1;cmd2;...}else{cmd3;cmd4;...}

# if-else多行格式
if(测试表达式){
        cmd1
        cmd2
        ...
}else{
        cmd3
        cmd4
        ...
}

# if-elseif-else单行格式
if(测试表达式){cmd1;cmd2;...}else if{cmd3;...}else{cmd7;...}

# if-elseif-else多行格式
if(测试表达式){
        cmd1
        cmd2
        ...
}
else if{
        cmd3
        cmd4
        ...
}
else{
        cmd5
        cmd6
        ...
```

(continues on next page)

(continued from previous page)

```
}

# 三目运算符
test_cmd ? cmd1 : cmd2
```

循环语句的格式有

```
# while循环多行格式
while(测试表达式){
        cmd1
        cmd2
        ....
}

# while循环单行格式
while(测试表达式){cmd1;cmd2;....}

# do循环多行格式
do{
        cmd1
        cmd2
}while(测试表达式)

# do循环单行格式
do{cmd1;cmd2}while(测试表达式)

# for循环多行格式
for(变量初始值; 测试表达式; 计数器增长表达式){
        cmd1
        cmd2
        ....
}

for(变量 in 数组){
        cmd1
        cmd2
        ....
}

# for循环单行格式
for(变量初始值; 测试表达式; 计数器增长表达式){cmd1;cmd2;....}

for(变量 in 数组){cmd1;cmd2;....}
```

需要说明的是：以上格式中所有cmd的本质就是一个语句或一个函数

在循环语句中有几个可以影响循环的动作

- break：退出循环。
- continue：退出当前循环，进入下一个循环
- next：读入下一行，awk程序的顶端从头开始，该语句只适用于PATTERN{action}这部分，不适用BEGIN{action}
- exit code：直接进入END，若本就在END中，则直接退出awk；如果END中的exit没有定义code，则采用前一个exit的code。

awk的ACTIONS中支持的函数有

- 输出函数

- 算术函数
- 字符串函数
- 自定义函数
- *system*函数
- *getline*函数

ACTIONS中使用print和printf函数输出数据，不仅可以输出到标准输出中，还可以重定向到文件中，甚至可以使用管道传递给另一个命令

```
# 输出数据到标准输出上
print                               # 将$0打印到标准输出,等价于print $0
print expression expression ... # 将各个expression的内容进行拼接然后打印到标准输出上,
由ORS终止
print expression,expression,... # 打印各个expression, expression之间由OFS 分开, 由ORS终
止
printf(format,expression,expression,...) # 格式化输出到标准输出上

# 输出数据重定向到文件
# 以下文件名filename必须使用双引号包围，否则被当作变量，且文件只会被打开一次
print expression,expression,... > filename  # 覆盖原文件内容
print expression,expression,... >> filename # 追加到原文件中
printf(format,expression,expression,...) > filename  # 格式化覆盖原文件内容
printf(format,expression,expression,...) >> filename # 格式化追加到原文件中

# 输出数据到另一个命令
# 以下命令需要使用双引号包围
print expression,expression,... | command  # 将数据传递给系统命令
printf(format,expression,expression,...) | command # 将格式化数据传递给系统命令
```

注意：如果print或printf的参数列表中含有操作符，则需要使用括号包围，否则容易产生歧义

```
print($1, $3) > ($3 > 100 ? "bigpop" : "smallpop")
print $1, ($2 > $3)
```

这里需要说明的是format格式化字串，它是一个包含输出格式说明符的纯文本字符串，输出格式说明符使用%来描述，然后跟着几个字符，这些字符控制一个value的输出格式。第一个%描述value1的输出格式，第二个%描述value2的输出格式，依次类推。因此，%的数量应该和被输出的value数量一样多

格式说明符%后可跟的字符有

- 格式符
  - %d：以十进制整数显示
  - %i：以十进制整数显示
  - %f：以浮点数显示
  - %s：以字符串显示
  - %u：以无符号整数显示
  - %%：显示%自身
  - %.：以小数值格式输出
- 修饰符
  - N：显示宽度；N为数值，宽度不足时若为左对齐则右边空格补足，若右对齐则左边空格补足
  - -：左对齐
  - +：显示数值正负号
  - 0：表示以0填充

例如：

```
printf("total pay for %s is $%.2f\n", $1, $2 * $3)
# 按字符串格式输出"$1"，按小数值格式输出"$2 * $3"，且小数位占2位

printf("%-8s $%6.2f\n", $1, $2 * $3)
# "%-8s"表示"$1"按字符串格式输出，但短横线"-"表示要左对齐输出，"8"表示占用8个字符宽度，不足之数在右边空格补齐
# "%6.2f"表示按小数格式输出"$2 * $3"，且小数位占用2位，总字符数占用6位。小数点也占用一个字符宽度。因此，一个可能的输出值为"123.20"
```

ACTIONS中支持的算术函数有

- cos(x)：取x的余弦
- sin(x)：取x的正弦
- sqrt(x)：取x的平方根
- rand()：返回一个随机数r，范围是[0,1)
- srand(x)：设置rand()的种子值为x。种子值相同时，rand()的结果相同。可print srand()输出当前种子值
- int(x)：取x的整数部分

ACTIONS中支持的字符串函数有(建议下面的所有regexp都使用//包围)

- index(str1,str2)：返回子串str2在字符串str1中第一次出现的位置。如果没有指定str1，则返回0
- length(str1)：返回字符串str1的长度。如果未给定str1，则表示计算"$0"的长度
- substr(str1,p)：返回str1中从p位置开始的后缀字符串
- substr(str1,p,n)：返回str1中从p位置开始，长度为n的子串
- match(str1,regexp)：如果regexp能匹配str1，则返回匹配起始位置。否则返回0。它会设置内置变量RSTART和RLENGTH的值
- split(str1,array,sep)：使用字段分隔符sep将str1分割到数组array中，并返回数组的元素个数。如果未指定sep则采用FS的值。因此该函数用于切分字段到数组中，下标从1开始
- sprintf(fmt,expr)：根据printf的格式fmt，返回格式化后的expr
- sub(regexp,rep,str2)：将str2中第一个被regexp匹配的字符串替换成rep，替换成功则返回1(表示替换了1次)，否则返回0，注意是贪婪匹配，替换字符串rep中使用&符号表示反向引用，引用的是整个被匹配的部分
- sub(regexp,rep)：将"$0"中第一个被regexp匹配的字符串替换成rep，替换成功则返回1，否则返回0，注意是贪婪匹配，替换字符串rep中使用&符号表示反向引用，引用的是整个被匹配的部分
- gsub(regexp,rep,str2)：将str2中所有被regexp匹配的内容替换成rep，并返回替换的次数，替换字符串rep中使用&符号表示反向引用，引用的是整个被匹配的部分
- gsub(regexp,rep)：将"$0"中所有被regexp匹配的内容替换成rep，并返回替换的次数，替换字符串rep中使用&符号表示反向引用，引用的是整个被匹配的部分
- toupper(str)：将str转换成大写字母，并返回新串
- tolower(str)：将str转换成小写字母，并返回新串

```
awk 'BEGIN{
        print index("banana","na")
        print length("banana")
        print match("banana","na.*")
        print toupper("banana")
        print substr("banana",3)}'
# 输出结果
```

(continues on next page)

(continued from previous page)

```
#3
#6
#3
#BANANA
#nana

awk 'BEGIN{str1="x&x";str2="banana"
        print sub(/a.*n/,str1,str2)
        print str2}'
# 输出结果
#1
#bxananxa

awk 'BEGIN{
        print match("banana",/a.*n/)
        print RSTART,RLENGTH}'
# 输出结果
#2
#2 4

awk 'BEGIN{print sprintf("hello %i world %5s","123","abc")}'
# 输出结果
#hello 123 world   abc

awk 'BEGIN{
        name="Ma long shuai"
        split(name,myname)
        for (i in myname){
                print myname[i]}
        }'
# 输出结果
#Ma
#long
#shuai

awk 'BEGIN{
        name="Ma:long:shuai"
        if (match(name,/:[^:]*:/)){
                print substr(name,RSTART+1,RLENGTH-2)}}'
# 输出结果(将匹配成功的字符串输出出来)
#long
```

自定义函数的格式如下

```
function name(parameter-list) {
        statements
}
```

自定义函数有以下特点

- 函数中的变量不影响函数外的变量，但可以使用外部变量。参数列表使用逗号分隔，这些参数只在函数内部生效
- 函数的定义可以在awk的引号内任意位置(即使是BEGIN之前或END之后)，但不能定义在BEGIN、主输入循环、END内部，否则自定义函数的大括号会和包围action的大括号冲突而报错
- 函数的调用位置可以在函数的定义位置之前
- 在函数的statements中，可以使用return expression语句，表示函数的返回值

如下**(1)-(4)**处位置可定义函数，可在任意位置处调用函数

```
awk '(1)BEGIN{ACTIONS}(2)PATTERN{ACTIONS}(3)END{ACTIONS}(4)'
```

以下实例创建了一个**向字符串指定位置处插入一个字符**的函数

```
awk 'function insert(STRING, POS, INS) {
        before_tmp = substr(STRING, 1, POS)
        after_tmp = substr(STRING, POS + 1)
        return before_tmp INS after_tmp
}
BEGIN{print insert("banana",3,"x")}'
```

system函数可以用来执行系统命令，但是命令需要使用引号包围，函数的返回值是命令的退出状态

```
awk 'BEGIN{system("fdisk -l")}'
awk 'BEGIN{name="ma long shuai";system("echo " name)}'
```

getline函数主要功能是：从文件、标准输入或管道中读取数据，并按情况设置变量的值。它可以自动不断的加载下一行

关于该函数的返回值有如下情况：

- 如果能读取记录，则返回值为1
- 如果遇到输入流的尾部，则返回值为0
- 如果不能读取记录(如文件没有读取权限、文件不存在)时，则返回值为-1

该函数有以下使用格式

- getline：会从主输入文件中读取记录，会同时设置$0,NF,NR,FNR
- getline var：会从主输入文件中读取记录，并将读取的记录赋值给变量var，同时会设置var,NR,FNR
- getline < file：从外部文件file中读取记录，同时会设置$0,NF，需要使用双引号包围文件名，否则被当成awk中的变量
- getline var < file：从外部文件file中读取记录，并将读取的记录赋值给变量var，需要使用双引号包围文件名，否则被当成awk中的变量
- cmd | getline：从管道中读取记录，会同时设置$0,NF
- cmd | getline var：从管道中读取记录，并将读取的记录赋值给变量var

也就是说

- 当getline从非主输入文件读取记录时，不会设置NR和FNR
- 当getline后没有给定变量var时，会将读取的记录赋值给$0，于是会同时设置NF并切分成字段；否则将读取的记录赋值给变量var，不会设置NF切分字段

例如

```
# 执行Linux下的who命令并传递给getline读取，每读取一行记录，变量n自增1
while ("who" | getline){n++}

# 将Linux命令date的结果保存到awk的变量date中
"date" | getline date

# 写成循环时，当无法读取file，返回值为"-1"，而while循环的判断条件是0和非0，所以"-1"会进入死循环
while (getline <"file" >0){cmd...}
```

0x0302 应用实例

```
awk 'BEGIN{name="ma long shuai";print (1,2,3,4) | "echo " name}'
awk 'BEGIN{while (("fdisk -l" | getline) >0){print $0}}'
awk 'BEGIN{system("fdisk -l")}'
awk 'BEGIN{name="ma long shuai";system("echo " name)}'
awk -F':' '$7 == "/bin/bash"{print "who use bash shell: ",$1}' /etc/passwd
awk 'BEGIN{print "ID NAME GENDER GENDER";print ""}{print $0}END{print "total num:
↪" NR}'
awk 'BEGIN{print rand();print rand();srand();print rand();print rand();print␣
↪srand()}'
```

文本编辑

目录

文本编辑器vim

vim是一款功能十分强大的文本编辑器，详细使用方法可参考：文本编辑器之vim

流式编辑器sed

参考文档

- sed从入门到深入的使用心得

学习sed的过程中，可以使用sedsed调试工具，这对于分析sed处理过程以及pattern space、hold space有很大帮助。

目录

sed基本使用

本文目录

0x00 sed本质

sed的本质就是一个流式编辑器。流式编辑器用于对输入流(文件、管道、标准输入传递的数据)执行基本的文本转换操作。它有以下特点(区别于其它类型编辑器):

- 能够筛选过滤管道传递过来的文本数据
- 只能通过一次输入流，即每次的输入流只能处理一次，因此它的效率更高。例如：(sed -n '2{p;q}';sed -n 3{p;q}) < filename命令中，第二个sed语句读取的输入流是空流

0x01 sed语法

sed命令的语法格式是：sed OPTIONS... [SCRIPT] [INPUTFILE...]

- OPTIONS：*sed*命令常用选项，注意当使用-e或-f script_file选项指定SCRIPT时，[SCRIPT]参数将会被屏蔽，即sed将会将其当做输入文件处理
- [SCRIPT]：*sed*命令脚本，[SCRIPT]是第一个非选项参数，sed仅在没有使用-e或-f script_file选项指定SCRIPT时，才将其当作是script部分而非输入文件
- [INPUTFILE...]：*sed*命令输入流，[INPUTFILE...]是第二个非选项参数，如果不指定INPUTFILE或者指定的INPUTFILE为-，sed将从标准输入中读取输入流并进行过滤

将上述sed语法格式可以扩写成以下几种语法(本质是对SCRIPT部分的展开)

```
# 没有使用-e选项的单个表达式==单行
sed OPTIONS... Address{cmd1;cmd2;cmd3...} [INPUTFILE...] # OPTIONS没有-e/f选
项，[SCRIPT]选项是Address{cmd1;cmd2;cmd3...}


# 使用-e选项的单个表达式==单行
sed OPTIONS... -e 'Address{cmd1;cmd2;cmd3...}' [INPUTFILE...] # 使用了OPTIONS中的-e选
项指定SCRIPT，此时[SCRIPT]选项被屏蔽省略


# 没有使用-e选项的多个表达式==单行，分号分隔
sed OPTIONS... Address1{cmd1;cmd2;cmd3};Address2{cmd1;cmd2;cmd3}... [INPUTFILE...]


# 使用-e选项的多个表达式==单行，空格分隔
sed OPTIONS... -e 'Address1{cmd1;cmd2;cmd3}' -e 'Address2{cmd1;cmd2;cmd3}' ...␣
↪[INPUTFILE...]


# 多个表达式==分行
sed OPTIONS... Address1{
        cmd1
        cmd2
        cmd3
}
Address2{
        cmd1
        cmd2
        cmd3
}
[INPUTFILE...]


# 使用-f选项指定SCRIPT脚本文件
sed OPTIONS... -f test.sed [INPUTFILE...] # 使用了OPTIONS中的-f选项指定SCRIPT脚本文
件，test.sed脚本文件的内容可以如下所示
#!/usr/bin/sed -f
#注释行
Address1{cmd1;cmd2...}
Address2{cmd1;cmd2...}
......
```

0x02 sed实现机制

sed实现机制可以使用下列编程结构来描述

```
for (line=1;line<=last_line_num;++line)
do
        read $line to pattern_space;
        while pattern_space is not null
        do
                if Address1;then execute cmd1 in SCRIPT;
                if Address2;then execute cmd2 in SCRIPT;
                if Address3;then execute cmd3 in SCRIPT;
                ......
                auto_print;
                remove_pattern_space;
        done
done
```

sed在对输入流进行处理时，维护了两个数据缓冲空间(这两个空间初始时都为空)

- 一直处于活动状态的模式空间(pattern space)
- 辅助性的保持空间(hold space)

实现sed机制的编程结构中包含两个循环

- *sed*循环：外层for循环
- *SCRIPT*循环：内层while循环

sed循环的处理流程是

- **(1)**sed读取输入流(文件、管道、标准输入)中的一行，移除该行的尾随换行符，并将其放入到模式空间(pattern space)中，同时将此行行号通过sed行号计数器记录在内存中
- **(2)**执行*SCRIPT*循环处理模式空间(pattern space)中读入的行内容
- **(3)**跳出*SCRIPT*循环后返回第一步操作，继续读取输入流的下一行，直到处理完输入流的最后一行才退出sed循环，sed处理完毕

SCRIPT循环的处理流程是

- **(1)**判断模式空间(pattern space)是否为空，如果为空，则跳出SCRIPT循环；如果不为空，就执行下面操作
- **(2)**如果模式空间(pattern space)中的内容能够匹配Address1，则执行Address1对应的cmd1对内容进行处理；否则不执行cmd1
- **(3)**如果模式空间(pattern space)中的内容能够匹配Address2，则执行Address2对应的cmd2对内容进行处理；否则不执行cmd2
- ...
- **(4)**当判断执行完所有的cmd后，auto_print会自动输出模式空间(pattern space)中的内容到标准输出流或定向流中，并添加尾随的换行符
- **(5)**最后remove_pattern_space会清空模式空间(pattern space)中的内容，并返回第一步操作

需要注意的是

- SCRIPT循环中的2/3步对应着我们*sed*脚本部分(即语法格式中的SCRIPT)，可以根据需要进行修改
- SCRIPT循环中的4/5步自动输出和清空模式空间内容，这两个操作每次SCRIPT循环都会默认自动执行，但是有些操作命令或选项可以改变这两个操作行为，使其输出总是输出空内容或无法输出或无法清空模式空间等

  - D命令会进入多行模式，使得SCRIPT循环结束时将数据锁在模式空间(pattern space)中不输出也不清空，并且在当前SCRIPT循环还没结束时就强行进入下一轮SCRIPT循环，其实就相当 于在上面的while循环结构中加上了continue关键字
  - d命令可以直接跳出SCRIPT循环进入下一个sed循环，就像是在while循环中加上了break一样
  - q和Q命令可以直接退出sed循环，就像是在while循环中加上了exit一样

0x03 sed常用选项

sed命令的常用OPTIONS选项有

- --help：输出sed命令行的简单帮助信息并退出，也可以使用info sed命令查看其帮助信息，info文档更全面详细点
- -n/--quiet/--silent：默认情况下，sed将在每轮*SCRIPT*循环结束时自动输出模式空间中的内容。使用该选项后可以使得这次自动输出动作输出空内容。注意，该选项是输出空内容而不是禁用输出动作，前者有输出流只是输出空流，后者则没有输出流；虽然两者的结果都是不输出任何内容，但在有些依赖于输出动作和输出流的地方，它们的区别是很大的。这种情况下，只有显式通过p命令来产生对应的输出
- -e SCRIPT/--expression=SCRIPT：*SCRIPT*是一个包含sed命令的表达式，-e选项就是向SCRIPT中添加命令的。可以省略-e选项，但如果命令行容易产生歧义，则使用-e选项可明确说明这部分是SCRIPT中的命令。另外，如果一个-e选项不方便描述所需命令集合时，可以指定多个-e选项
- -f SCRIPT-FILE/--file=SCRIPT-FILE：指定包含sed命令集合的SCRIPT文件(即将-e选项后面的SCRIPT表达式写入文件中)，让sed根据SCRIPT文件中的命令集处理输入流
- -i[SUFFIX]/--in-place[=SUFFIX]
  - 该选项指定要将sed的输出结果保存(以覆盖的方式)到当前编辑的文件中。该项是通过创建一个临时文件并将输出写入到该临时文件，最后重命名为源文件来实现的。该选项隐含了-s选项
  - 当处理完当前输入流后，临时文件被重命名为源文件的名称，如果没有提供SUFFIX，源文件被覆盖，且不会生成备份文件；如果提供了SUFFIX，则在重命名临时文件之前，先使用该SUFFIX修改源文件名，从而生成一个备份文件(例如sed -i'.log' SCRIPT a.txt将生成两个文件a.txt和a.txt.log，前者是sed修改后的文件，a.txt.log是源a.txt的备份文件)
  - 注意：如果SUFFIX不包含符号*，将SUFFIX添加到原文件名的后面当作备份文件的后缀；如果SUFFIX中包含了一个或多个字符*，则每个*都替换为原文件名；这使得可以为备份文件添加一个前缀，而不是后缀，甚至可以将此备份文件放在在其他已存在的目录下
- -r/--regexp-extended：使用扩展正则表达式，而不是使用默认的基础正则表达式；sed所支持的扩展正则表达式和egrep一样，使用扩展正则表达式显得更简洁，因为有些元字符不用再使用反斜线\，但这是GNU扩展功能，因此应避免在可移植性脚本中使用
- -s/--separate：默认情况下，如果为sed指定了多个输入文件，例如sed OPTIONS SCRIPT file1 file2 file3，则多个文件会被sed当作一个长的输入流处理，也就是说所有文件被当成一个大文件进行处理。指定该选项后，sed将认为命令行中给定的每个文件都是独立的输入流；既然是独立的输入流，范围定界(如/abc/,/def/)就无法跨越多个文件进行匹配，行号也会在处理每个文件时重置，$代表的也将是每个文件的最后一行
- -l N/--line-length=N：为l命令指定默认的换行长度。N=0意味着完全不换行的长行，如果不指定，则70个字符就换行
- --follow-symlinks：该选项只在支持符号连接的操作系统上生效，且只有指定了-i选项时才生效。指定该选项后，如果sed命令行中指定的输入文件是一个符号连接，则sed将对该符号链接的目标文件进行处理。默认情况下，禁用该选项，因此不会修改链接的源文件
- -u/--unbuffered:使用尽量少的空间缓冲输入和输出行；该选项在某些情况下尤为有用，例如输入流的来源是tail -f时，指定该选项将可以尽快返回输出结果

- `-z/--null-data/--zero-terminated`：以空串符号`\0`而不是换行符`\n`作为输入流的行分隔符

0x04 sed程序

0x0400 sed程序之范围定界

0x0401 sed程序之操作命令

0x05 sed脚本

0x06 sed输入流

0x06 sed应用

sed高级应用

sed常见问题

IO操作

命令汇总

- *read*
- *echo*
- *printf*

0x00 read

参考文档：shell脚本之read命令

`read`命令是用来获取用户输入内容，即标准输入设备(键盘)输入内容，它是shell内建命令，使用`help read`命令可以查看其语法格式和使用说明，它的语法格式如下

```
read [-ers] [-a array] [-d delim] [-i text] [-n nchars] [-N nchars] [-p prompt] [-
↪t timeout] [-u fd] [var_name ...]
```

其执行逻辑如下

- `read`命令从标准输入设备中读取输入单行，默认单行的结束符号为回车换行符
  - 此处需要注意的是：不带任何选项的read命令，只有按下回车键才能结束read命令的读取
- 然后将读取的单行根据`IFS`环境变量分裂成多个字段，并将分割后的字段分别赋值给`read`命令后面指定的变量列表`var_name`，其赋值逻辑如下
  - 第一个字段分配给第一个变量`var_name1`，第二个字段分配给第二个变量`var_name2`，依次到结束
  - 如果指定的变量名少于字段数量，则多出的字段数量也同样分配给最后一个`var_name`
  - 如果指定的变量命令多于字段数量，则多出的变量赋值为空
  - 如果没有指定任何`var_name`，则分割后的所有字段都存储在特定变量`REPLY`中

read命令的常用选项有

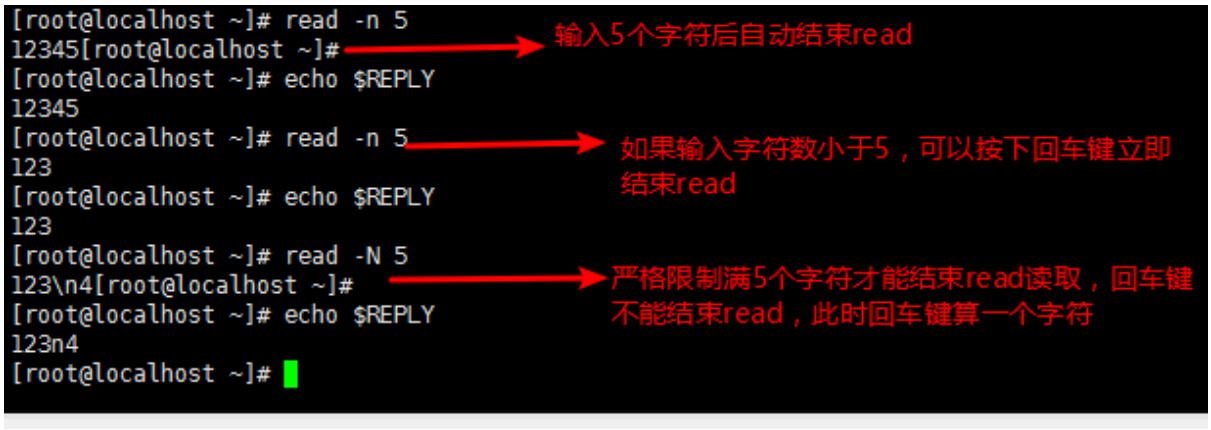
- -a：将分隔后的字段依次存储到-a指定的数组中，存储的起始位置从数组的index=0开始
- -d：指定读取行的结束符号，默认结束符号为换行符
- -n：限制读取N个字符就自动结束读取，如果没有读满N个字符就按下回车或遇到换行符，则也会结束读取
- -N：严格要求读满N个字符才自动结束读取，即使中途按下了回车或遇到了换行符也不结束，其中换行符或回车算一个字符
- -p：输出提示符或提示语，默认不支持\n换行，要换行需要特殊处理
- -r：禁止反斜线的转义功能，这意味着\会变成文本的一部分
- -s：静默模式，输入的内容不会回显在屏幕上，常用来获取密码输入
- -t：给出超时时间，在达到超时时间时，read退出并返回错误，也就是说不会读取任何内容，即使已经输入了一部分

-a选项将读取的内容分配给数组变量，从索引号0开始分配

```
[root@localhost ~]# read -a array_test
my name is anony
[root@localhost ~]# echo ${array_test[@]}
my name is anony
[root@localhost ~]# echo ${array_test[0]}
my
[root@localhost ~]#
```

-d选项指定读取行的结束符号，而不再使用换行符

-n和-N选项限制输入字符



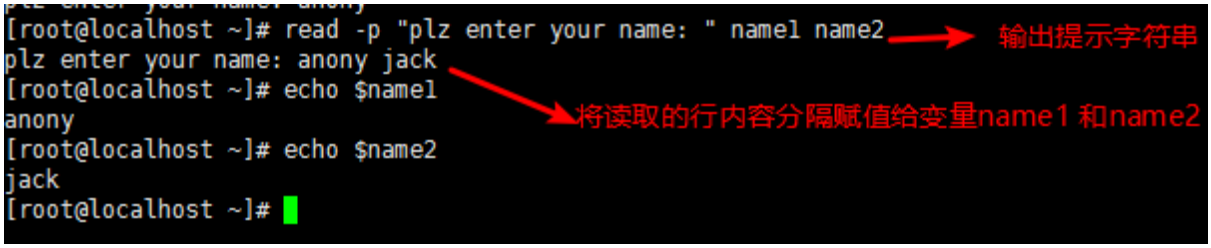
-p选项输出提示字符串

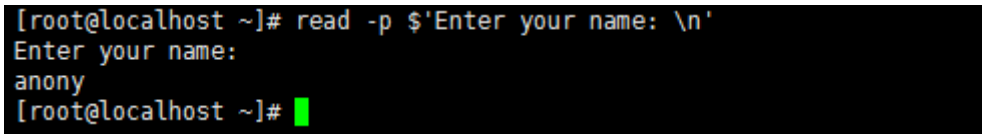
-p选项默认不带换行功能，且也不支持\n换行，但通过$'string'的方式特殊处理，就可以实现换行的功能；关于$'String'和$"String"的作用，详见shell中加引号有什么用

-s选项用来获取密码输入

-t选项给出输入时间限制，没完成的输入将被丢弃，所有变量将赋值为空(如果在执行read前，变量已被赋值，则此变量在read超时后将被覆盖为空)

read也可以用来在shell脚本中读取文件内容






```
# 每读取文件一行内容，就会进入一次while循环，直到读完文件尾部退出循环

# 读取文件方法一
while read line; do
        echo $line
done < /etc/passwd

# 读取文件方法二
exec </etc/passwd;while read line; do
echo $line
done
```

0x01 echo

echo命令类似于c中printf，用于标准输出，它是shell内建命令，使用help read命令可以查看其语法格式和使用说明，它的语法格式如下

```
echo [-neE] [arg ...]
```

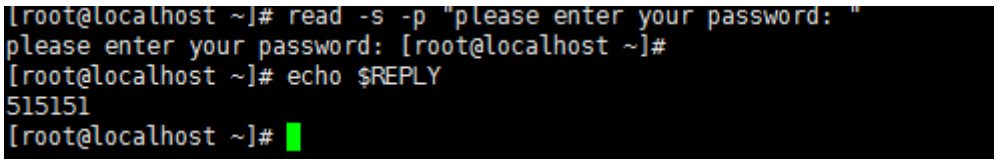
它的执行逻辑是：将给定arg内容按照-neE选项指定的不同方式输出

echo命令常用的选项有

- -n：取消分行输出
- -e：支持字符串内转义字符的显示输出

关于echo命令的使用，主要关注一些几点

- echo中的引号和感叹号：在bash环境中，感叹号只能通过单引号包围来输出，不能通过双引号来包围输出，原因有
  - 在bash环境中，感叹号表示引用历史命令，除非设置set +H关闭历史命令的引用
  - ''：单引号表示强引用，该操作符的优先级大于!，即不会进行历史命令的引用，直接引用显示全部字符
  - ""：双引号表示弱引用，该操作符的优先级小于!，即先进行历史命令的引用，然后再引用显示全部字符
- echo中的转义：通过-e选项识别转义和特殊意义的符号，如换行符\n、制表符\t、转义符\等



```
[root@localhost ~]# var=5
[root@localhost ~]# read -t 3 var
1[root@localhost ~]# 1
-bash: 1: command not found
[root@localhost ~]# echo $var

[root@localhost ~]#
```

```
[root@localhost ~]# echo hello world!
hello world!
[root@localhost ~]# echo 'hello world!'
hello world!
[root@localhost ~]# echo "hello world!"
-bash: !": event not found
[root@localhost ~]# echo hello world!;echo 'hello world!'
-bash: !: event not found
[root@localhost ~]# echo 'hello world!';echo hello world!
hello world!
hello world!
[root@localhost ~]# set +H
[root@localhost ~]# echo "hello world!"
hello world!
[root@localhost ~]#
```

默认双引号不能包围！

！只能在最尾部部

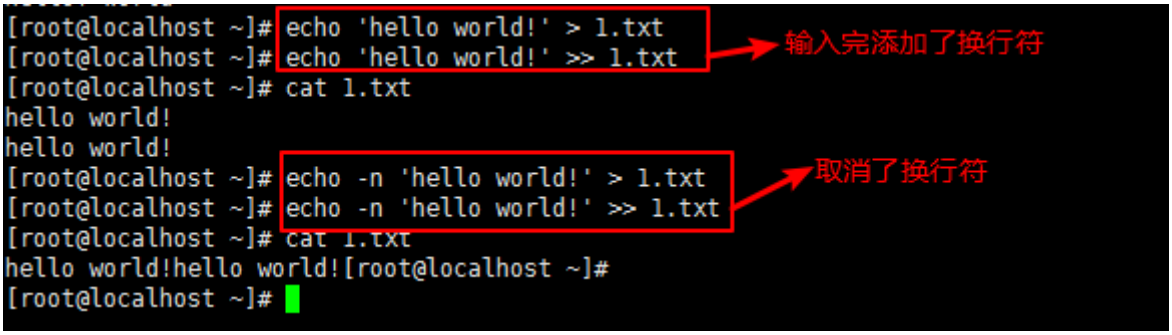
取消！的历史命令引用功能

```
echo "hello world"    # 打印字符串

# -e选项支持字符串内转义字符的显示输出
echo -e "hello\bworld"          # 删除前面的字符，输出hellworld
echo -e "hello\tworld"          # 制表符，输出hello     world
echo -e "hello\vworld"          # 垂直制表符
echo -e "hello\nworld"          # 换行符
```

- echo中的分行处理：默认情况下echo会在每行行尾加上换行符号，使用-n选项可以取消分行输出
- echo中的颜色输出：echo可以控制字体颜色和背景颜色输出，因为需要使用特殊符号，所以需要配合-e选项来识别特殊符号
  - 常见的字体颜色：重置=0，黑色=30，红色=31，绿色=32，黄色=33，蓝色=34，紫色=35，天蓝色=36，白色=37
  - 常见的背景颜色：重置=0，黑色=40，红色=41，绿色=42，黄色=43，蓝色=44，紫色=45，天蓝色=46，白色=47
  - 字体控制选项：1表示高亮，4表示下划线，5表示闪烁
  - 着色显示字符串格式为："\033[@;@mSTRING\033[0;0m"从左往右各字段的含义依次是
    - \033表示定义一个转义序列，也可以使用\e
    - [表示开始定义颜色
    - @;@表示颜色定义，第一个@表示字背景颜色，颜色范围40–47；;用来分隔字背景颜色和文字颜色；第二个@表示文字颜色，颜色范围30–37。如果没有相关定义则表示默认颜色
    - m表示颜色定义完毕

```
[root@localhost ~]# str=hello
[root@localhost ~]# echo "$str"'!' "world"
hello! world
[root@localhost ~]#
```

只用单引号无法扩展变量，使用双引号不好输出感叹号，于是解决方法就是：对这种特殊符号分开引用



* STRING表示要输出的字符串
* \033表示定义一个转义序列，也可以使用\e
* [表示再次开启颜色定义
* 0;0m表示将前面定义的背景颜色和文字颜色重置为默认颜色；注意定义了颜色之后就需要使用此项来重置关闭颜色，否则会继续影响bash环境的颜色，前面定义了几个@，该处就应该使用几个0来重置对应的颜色

```
echo -e "\033[32mhello\033[0m"          # 着色显示，默认背景颜色，字颜色为32绿色
echo -e "\033[34m 蓝色字 \033[0m"
echo -e "\033[35m 紫色字 \033[0m"
echo -e "\033[36m 天蓝字 \033[0m"
echo -e "\033[37m 白色字 \033[0m"
echo -e "\033[40;37m 黑底白字 \033[0m"
echo -e "\033[41;37m 红底白字 \033[0m"
echo -e "\033[42;37m 绿底白字 \033[0m"
echo -e "\033[43;37m 黄底白字 \033[0m"
echo -e "\033[44;37m 蓝底白字 \033[0m"
echo -e "\033[45;37m 紫底白字 \033[0m"
echo -e "\033[46;37m 天蓝底白字 \033[0m"
echo -e "\033[47;30m 白底黑字 \033[0m"
echo -e "\033[41;37;0m 关闭所有属性 \033[0m"
echo -e "\033[41;37;1m 设置高亮度 \033[0m"
echo -e "\033[41;37;4m 下划线 \033[0m"
echo -e "\033[41;37;5m 闪烁 \033[0m"
echo -e "\033[41;37;7m 反显 \033[0m"
echo -e "\033[41;37;8m 消隐 \033[0m"
```

0x02 printf

使用printf命令可以输出比echo更规则更格式化的结果，它引用于C语言的printf函数，但是有些许区别；它也是shell内建命令，使用help printf命令可以查看其语法格式和使用说明，它的语法格式如下

```
printf [-v var] format [arguments]
```

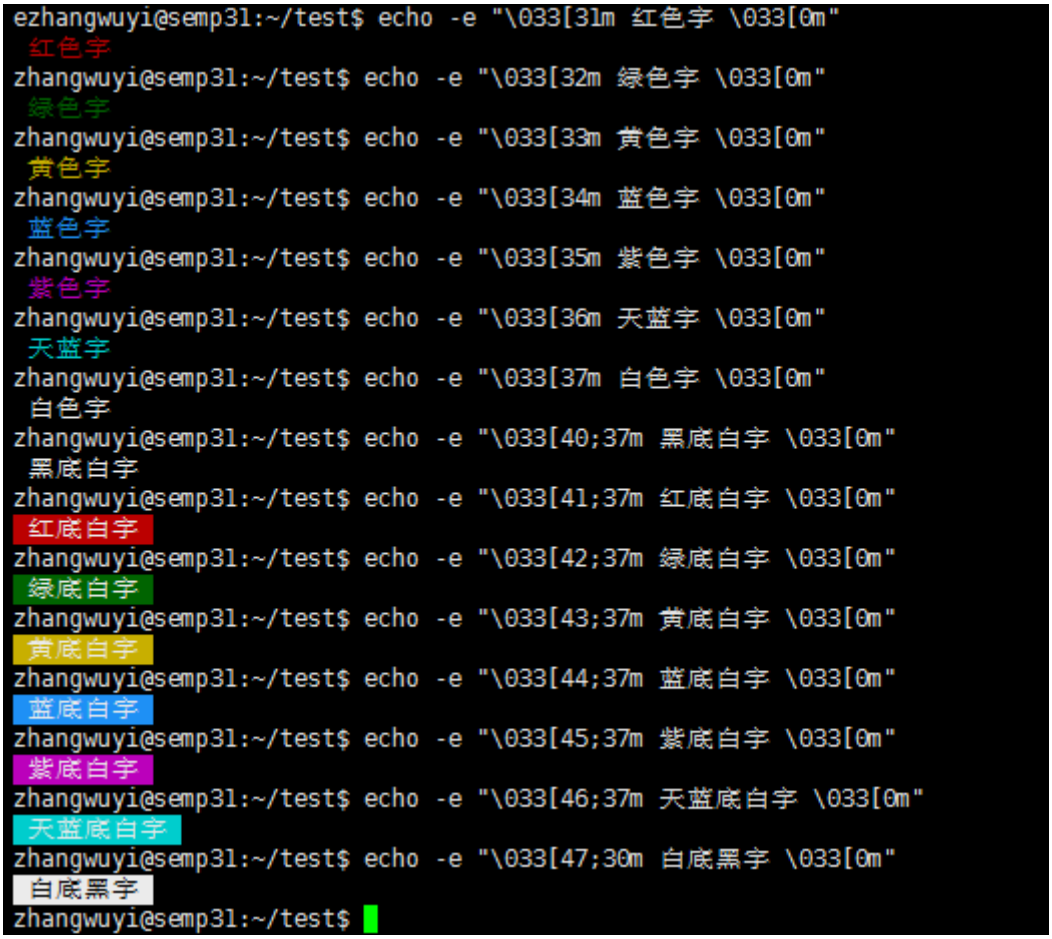
其执行逻辑是：按照format定义的输出格式将arguments输出到指定位置；默认是输出到标准输出，如果使用了-v选项表示将arguments按照指定格式赋值给该选项指定的变量var

使用printf最需要注意以下两点

- printf默认不在结尾加换行符，它不像echo一样，所以要手动加\n换号符
- printf只是格式化输出，不会改变任何结果，所以在格式化浮点数的输出时，浮点数结果是不变的，仅仅只是改变了显示的结果

使用printf可以实现

- 指定字符串的宽度


ezhangwuyi@semp31:~/test$ echo -e "\033[31m 红色字 \033[0m"
红色字
zhangwuyi@semp31:~/test$ echo -e "\033[32m 绿色字 \033[0m"
绿色字
zhangwuyi@semp31:~/test$ echo -e "\033[33m 黄色字 \033[0m"
黄色字
zhangwuyi@semp31:~/test$ echo -e "\033[34m 蓝色字 \033[0m"
蓝色字
zhangwuyi@semp31:~/test$ echo -e "\033[35m 紫色字 \033[0m"
紫色字
zhangwuyi@semp31:~/test$ echo -e "\033[36m 天蓝字 \033[0m"
天蓝字
zhangwuyi@semp31:~/test$ echo -e "\033[37m 白色字 \033[0m"
白色字
zhangwuyi@semp31:~/test$ echo -e "\033[40;37m 黑底白字 \033[0m"
黑底白字
zhangwuyi@semp31:~/test$ echo -e "\033[41;37m 红底白字 \033[0m"
红底白字
zhangwuyi@semp31:~/test$ echo -e "\033[42;37m 绿底白字 \033[0m"
绿底白字
zhangwuyi@semp31:~/test$ echo -e "\033[43;37m 黄底白字 \033[0m"
黄底白字
zhangwuyi@semp31:~/test$ echo -e "\033[44;37m 蓝底白字 \033[0m"
蓝底白字
zhangwuyi@semp31:~/test$ echo -e "\033[45;37m 紫底白字 \033[0m"
紫底白字
zhangwuyi@semp31:~/test$ echo -e "\033[46;37m 天蓝底白字 \033[0m"
天蓝底白字
zhangwuyi@semp31:~/test$ echo -e "\033[47;30m 白底黑字 \033[0m"
白底黑字
zhangwuyi@semp31:~/test$


zhangwuyi@semp31:~/test$ echo -e "\033[41;37;0m 关闭所有属性 \033[0m"
关闭所有属性
"zhangwuyi@semp31:~/test$ echo -e "\033[41;37;1m 设置高亮度 \033[0m"
设置高亮度
zhangwuyi@semp31:~/test$ echo -e "\033[41;37;4m 下划线 \033[0m"
下划线
zhangwuyi@semp31:~/test$ echo -e "\033[41;37;5m 闪烁 \033[0m"
闪烁
zhangwuyi@semp31:~/test$ echo -e "\033[41;37;7m 反显 \033[0m"
反显
zhangwuyi@semp31:~/test$ echo -e "\033[41;37;8m 消隐 \033[0m"
zhangwuyi@semp31:~/test$

- 实现左对齐(使用-)
- 实现右对齐(默认值)
- 格式化小数输出

```
#!/bin/bash

# 三个%分别对应后面的三个参数
# 减号"-"表示左对齐，默认表示右对齐
# 减号"-"后面的数字n表示占用n个字符
# 点号"."后面的数字m表示取小数点后m位
# s表示对应一个字符串变量
# f表示对应一个浮点数变量
# d表示对应一个整数变量
# \t表示制表符
# \n表示换行符
printf "%-s\t %-s\t %s\n" No Name Mark
printf "%-s\t %-s\t %4.2f\n" 1 Sarath 80.34
printf "%-s\t %-s\t %4.2f\n" 2 James 90.998
printf "%-s\t %-s\t %4.2f\n" 3 Jeff 77.564

# 执行结果如下
# No              Name            Mark
# 1               Sarath          80.34
# 2               James           91.00
# 3               Jeff            77.56
```

用户管理

useradd

adduser

软件包管理

进程管理

目录

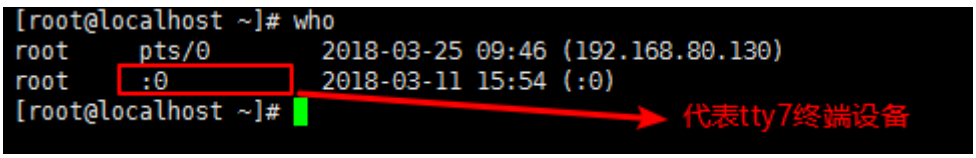
基本操作

命令汇总

- who

who

who命令可以用来查看当前在线用户的情况



每个字段的意义分别是：

- 登陆的用户名
- 使用的设备终端
- 登陆系统的时间

在linux中我们的终端设备有：

- pts设备终端
- tty设备终端
  - tty1~tty6：表示文字界面，在shell下敲`ctrl+alt+[F1-F6]`即可进入对应的文字界面
  - tty7：表示图像界面，在shell下敲`ctrl+alt+F7`即可进入图形界面

ps

`ps`命令是用来查看整个系统内部运行进程的相关信息

```
$ps -a          # 列出当前登录终端的进程信息
$ps -au         # 列出当前登录终端以及对应用户的信息
$ps -aux        # 列出当前没有对应终端的进程信息，没有终端就意味着不能和用户进行交互

# ps命令的显示结果一般都很杂乱，不容易找到我们需要的信息，所以ps命令一般通过管道和grep一起使用
$ps -aux | grep -e "^root.*httpd$" # 查找进程属主为root，进程名为httpd的进程相关信息
```

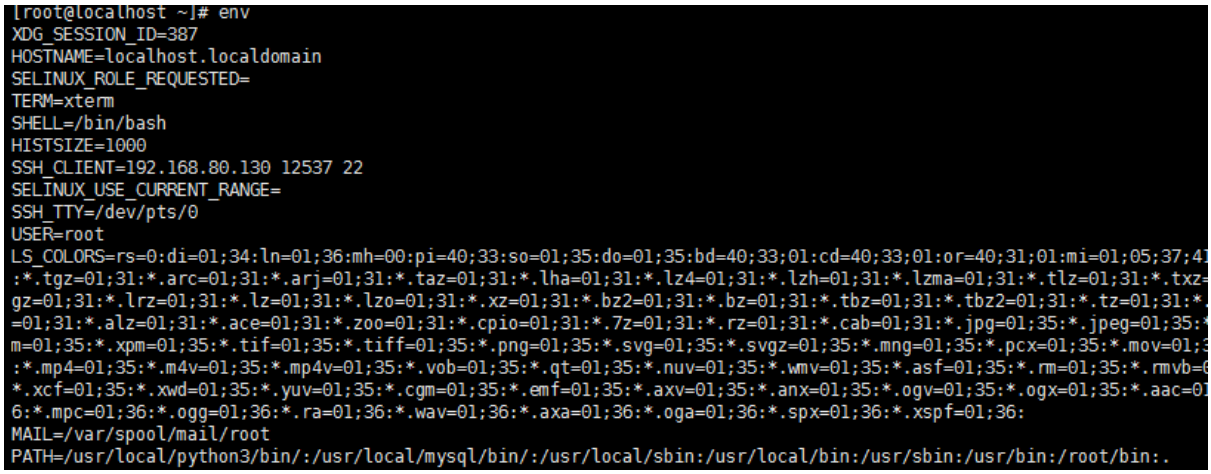
kill

`kill`命令是通过发送信号来杀死运行进程的

```
$kill -l              # 查看当前系统下可以使用的信号
$kill -SIGKILL 5179 # 使用SIGKILL信号杀死进程号为5179的进程
$kill -9 4968         # 使用SIGKILL信号杀死进程号为4968的进程，-9对应SIGKILL信号
```

env

`env`命令是用来查看当前进程的环境变量



如上图所示，linux下环境变量的格式是：`key=value`

- `key`一般都是大写的变量
- `value`值可以有多个，每个值之间使用`:`分隔

```
$env | grep PATH  $获取当前进程的PATH环境变量
```

top

top命令可以实时刷新进程的状态信息，类似于windows下面的任务管理器

init进程是内核启动后，在用户空间中启动的第一个进程，由此进程生成其他需要的子进程，例如：用于交互启动其他用户程序进程的shell进程就是init进程的子进程。但凡涉及到用户空间中的进程，内核将不再管理，直接由init进程来管理

IPC进程通信

shell命令中有以下方式实现进程(两个命令)之间通信

- 管道
- *xargs*
- *exec*

0x00 管道

参考文档:

- Linux管道的实现机制
- linux管道pipe详解

0x01 xargs

参考文档:

- xargs百度百科
- xargs命令详解
- xargs的原理剖析及用法详解

0x02 exec

参考文档

- find命令之exec

系统管理

网络管理

ifconfig

ping

ping命令是用来测试与目标主机的连通性

nslookup

nslookup命令是用来查看服务器域名对应的IP地址

服务管理

目录

服务器搭建

目录

ftp服务器

参考文档

- CentOS下vsftp设置
- 针对CentOS的SELinux拦截vsftpd问题

ftp服务器的主要功能就是：文件的上传和下载

ftp服务器的搭建我们这里选择使用vsftpd工具

其搭建基本流程是：

- *vsftpd*工具安装
  - *Ubuntu*系列安装
  - *CentOS*系列安装
- 服务端配置
  - 修改配置文件
  - 重启服务
- 客户端登陆
  - *ftp*客户端访问
  - *lftp*客户端访问
- 连接操作
  - *ftp*客户端操作
  - *lftp*客户端操作

0x00 vsftpd工具安装

不同linux发行版的安装方式不同

Debian/Ubuntu系列安装方式

```
# apt-get安装
$sudo apt-get install vsftpd
```

RedHat/Fedra/CentOs系列安装方式

```
# yum安装
$yum install vsftpd
```

0x01 服务端配置

安装好vsftpd工具之后，我们需要修改下配置文件，但是我们怎么知道配置在哪呢，如果熟悉linux的人就会知道配置文件肯定在/etc目录下，但是该目录那么多配置文件我该怎么找呢？此时我们有几种方式可以帮你找到其配置文件

```
$ rpm -ql vsftpd | grep etc    # 获取vsftpd工具配置文件的安装路径
$ whereis vsftpd               # 获取vsftpd工具配置文件的安装路径
```

通过上述命令我们可以得知，其配置文件路径是：/etc/vsftpd/vsftpd.conf，通过vi/vim编辑器打开，具体修改内容如下：

```
21 #
22 # Allow anonymous FTP? (Disabled by default)
23 anonymous_enable=NO
24 #
25 # Uncomment this to allow local users to log in.
26 local_enable=YES
27 #
28 # Uncomment this to enable any form of FTP write command.
29 #write_enable=YES
30 #
31 # Default umask for local users is 077. You may wish to change this to 022,
32 # if your users expect that (022 is used by most other ftpd's)
33 #local_umask=022
34 #
35 # Uncomment this to allow the anonymous FTP user to upload files. This only
36 # has an effect if the above global write enable is activated. Also, you will
37 # obviously need to create a directory writable by the FTP user.
38 #anon_upload_enable=YES
39 #
40 # Uncomment this if you want the anonymous FTP user to be able to create
41 # new directories.
42 #anon_mkdir_write_enable=YES
43 #
44 # Activate directory messages - messages given to remote users when they
45 # go into a certain directory.
46 dirmessage_enable=YES
```

anonymous_enable=YES　允许匿名用户登录

#write_enable=YES --> write_enable=YES　实名登录用户拥有写权限（上传数据）

#local_umask=022 --> local_umask=022　设置本地掩码为 022

#anon_upload_enable=YES --> anon_upload_enable=YES　匿名用户可以向ftp服务器上传数据

去掉该行前边的#　匿名用户可以在ftp服务器上创建目录

配置文件修改完之后，需要重启服务使其配置文件生效

```
# Systemv格式重启服务
$service vsftpd restart            # CentOS系列重启
$sudo service vsftpd restart       # Ubuntu系列重启

# Systemd格式重启服务
$systemctl restart vsftpd.service        # CentOS系列重启
$sudo systemctl restart vsftpd.service   # Ubuntu系列重启
```

最后使用netstat -pantu | grep vsftpd命令查看服务是否启动成功，处于监听状态

```
[root@localhost ~]# netstat -pantu | grep vsftpd
tcp6       0      0 :::21                   :::*                    LISTEN      1194/vsftpd
[root@localhost ~]#
```

0x02 客户端登陆

使用ftp客户端登陆之前，我们需要安装ftp客户端

- linux端
  - yum install ftp
  - sudo apt-get install ftp

- windows端
  - Xmanage-Xshell
  - Securecrt

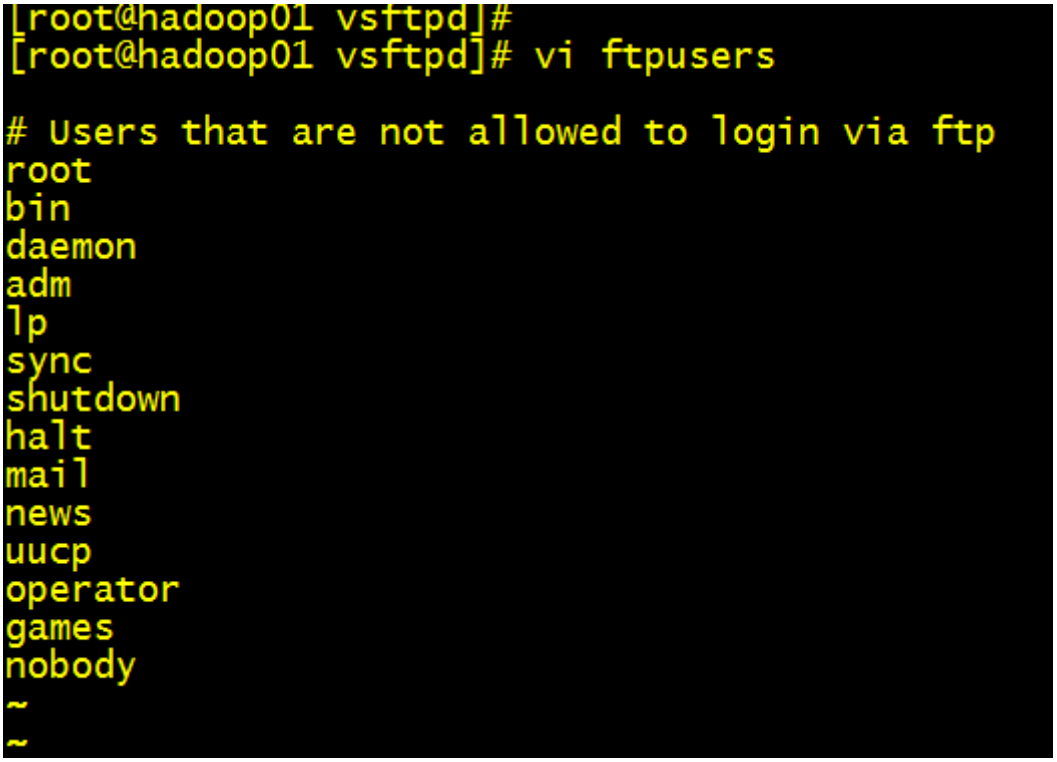
安装好客户端后就可以使用ftp客户端登陆了，登陆方式有：

- 实名用户登陆
- 匿名用户登陆

实名用户登陆(登陆默认进入登陆用户的家目录):

```
# linux下使用ftp登陆
$ftp 192.168.80.128          # 然后输入用户名和密码即可

# windows下xshell登陆
$ftp zwy@192.168.80.128      # zwy是要登陆的用户，然后输入密码即可
```

注意：有些用户可能不具有ftp的权限，这些用户会存在于/etc/vsftpd/ftpusers 和/etc/vsftpd/user_list文件中。如果想让这些用户可以ftp登陆，则需要将其在这两个文件删除或注释掉即可



实名登陆有诸多弊端:

- 暴露用户名和密码
- 实名用户登陆后可以在任意目录下来回切换，不太安全

所以一般我们都不使用实名用户登陆，而是使用匿名用户进行登陆

匿名用户登陆后，是不允许匿名用户在任意目录下直接来回切换，只能在一个指定目录范围内工作，所以需要在ftp服务器上创建一个匿名用户的家目录，该目录就是匿名用户的根目录

匿名用户是以ftp用户登陆的(默认无密码)，所以默认匿名用户的家目录就是ftp用户的家目录，此时我们可以通过cat /etc/passwd | grep ftp命令找到ftp用户的家目录，也就是匿名用户的家目录

然后使用下列方法实现匿名用户登陆

```
# linux下使用ftp登陆
$ftp 192.168.80.128          # 然后输入用户名anonymous和密码，密码直接回车即可

# windows下xshell登陆
$ftp anonymous@192.168.80.128     # anonymous是匿名用户，然后输入密码，密码直接回车即可
```

除了可以使用默认ftp家目录作为匿名用户根目录，还可以自己设定

- 使用mkdir anonyFtp命令在指定目录下创建一个目录，即为匿名用户的家目录
- 在vsftpd.conf配置文件中通过‘‘ anon_root=/home/anonyFtp‘‘设定该目录为匿名用户的登陆根目录
- anonyFtp目录的所有者模型可以使用默认的owner、group所有者，但是权限模型最好设置为777，即chmod 777 anonyFtp。此时owner和group都不是ftp
  - 如果要保证匿名用户能够登陆，必须保证others所有者权限位+x
  - 如果要保证能够列出根目录下所有文件内容，必须保证others所有者权限位+r
  - 如果要保证能够在根目录上上传文件，必须保证others所有者权限位+w
- 如果要保证anonyFtp根目录下所有子目录能否切换、上传、下载，设置方法如下，这样可以保证ftp用户有可执行x权限
  - chown ftp:ftp anonyFtp/pub：将pub目录的owner和group都设置为ftp，因为匿名用户是以ftp用户登陆的
  - chmod 744 anonyFtp/pub
- vsftpd默认被CentOS的防火墙组件SELinux拦截，造成vsftpd没有足够的权限，有两种解决方法
  - 直接关闭SELinux，这样不太安全
    - setenforce 0：暂时让SELinux进入Permissive模式，关闭SELinux，但是重启失效
    - 将/etc/selinux/config文件中的SELINUX=enforcing改成SELINUX=permissive,然后重读该配置文件，重启也生效
  - 不需要关闭SELinux就能使vsftpd具有访问ftp根目录，以及文件传输等权限
    - getsebool -a | grep ftpd：查看与ftpd相关的权限信息，off是关闭权限，on是打开权限
    - setsebool -P ftpd_anon_write 1：打开ftpd_anon_write权限，重启生效
    - setsebool -P allow_ftpd_full_access 1：打开allow_ftpd_full_access权限，重启生效
- 最后重启服务

使用lftp客户端登陆之前，我们需要安装lftp客户端

- linux端
  - yum install lftp
  - sudo apt-get install lftp

安装好客户端后就可以使用lftp客户端登陆了，登陆方式同样也有：

- 实名用户登陆
- 匿名用户登陆

匿名用户登陆步骤：

- lftp 192.168.80.128，回车
- 输入login，回车

如果使用匿名用户登陆后ls出现如下错误提示：refusing to run with writable root inside chroot()，是因为默认配置中不允许根目录可写，有两种解决方法：

- 方法一：chmod o-w anonyFtp：去掉根目录的可写权限
- 方法二：在/etc/vsftpd/vsftpd.conf配置文件中，添加allow_writeable_chroot=YES字段

实名用户登陆步骤：

- lftp test@192.168.80.128，回车
- 输入密码回车

0x03 连接操作

ftp客户端登陆后输入help即可查看ftp支持的所有操作命令

linux中ftp工具支持的操作

```
ftp> help
Commands may be abbreviated.  Commands are:

!               debug           mdir            sendport        site
$               dir             mget            put             size
account         disconnect      mkdir           pwd             status
append          exit            mls             quit            struct
ascii           form            mode            quote           system
bell            get             modtime         recv            sunique
binary          glob            mput            reget           tenex
bye             hash            newer           rstatus         tick
case            help            nmap            rhelp           trace
cd              idle            nlist           rename          type
cdup            image           ntrans          reset           user
chmod           lcd             open            restart         umask
close           ls              prompt          rmdir           verbose
cr              macdef          passive         runique         ?
delete          mdelete         proxy           send
ftp>
```

windows中xshell工具支持的操作

注意：ftp客户端不能直接操作目录，如果要上传下载目录，需要先将其打包压缩成一个文件

lftp客户端登陆后输入help即可查看lftp支持的所有操作命令

其中常用操作有：

- put：上传文件
- mput：上传多个文件
- get：下载文件
- mget：下载多个文件
- mirror：下载整个目录以及子目录
- mirror -R：上传整个目录以及子目录

由上述可知：lftp客户端既可以操作文件，又可以直接操作目录

nfs服务器

参考文档：

```
ftp:/home/zhangwuyi> help
ascii    set ascii transfer type
bin      set binary transfer type
binary   set binary transfer type
cd       change remote working directory
cdup     change remote working directory to parent directory
delete   delete remote file
dir      list contents of remote directory
exit     terminate ftp session and exit
get      receive file
help     print local help information
lcd      change local working directory
ldir     list contents of local directory
lls      list contents of local directory
lpwd     print working directory on local machine
ls       list contents of remote directory
mget     get multiple files
mkdir    make directory on the remote machine
mput     put multiple files
mv       rename file
passive  toggle pasive transfer mode
put      send one file
quit     terminate ftp session and exit
quote    send arbitrary ftp command
rename   rename file
rm       delete remote file
rmdir    remove directory on the remote machine
user     send new user information
ftp:/home/zhangwuyi>
```

```
[root@localhost ~]# lftp 192.168.80.128
lftp 192.168.80.128:~> help
    !<shell-command>                  (commands)                        alias [<name> [<value>]]          attach [PID]                      bookmark [SUBCMD]
    cache [SUBCMD]                    cat [-b] <files>                  cd <rdir>                         chmod [OPTS] mode file...         close [-a]
    [re]cls [opts] [path/][pattern]   debug [<level>|off] [-o <file>]   du [options] <dirs>               exit [<code>|bg]                  get [OPTS] <rfile> [-o <lfile>]
    glob [OPTS] <cmd> <args>          help [<cmd>]                      history -w file|-r file|-c|-l [cnt] jobs [-v] [<job_no...>]          kill all|<job_no>
    lcd <ldir>                        lftp [OPTS] <site>                ln [-s] <file1> <file2>           ls [<args>]                       mget [OPTS] <files>
    mirror [OPTS] [remote [local]]    mkdir [-p] <dirs>                 module name [args]                more <files>                      mput [OPTS] <files>
    mrm <files>                       mv <file1> <file2>                [re]nlist [<args>]                open [OPTS] <site>                pget [OPTS] <rfile> [-o <lfile>]
    put [OPTS] <lfile> [-o <rfile>]   pwd [-p]                          queue [OPTS] [<cmd>]              quote <cmd>                       repeat [OPTS] [delay] [command]
    rm [-r] [-f] <files>              rmdir [-f] <dirs>                 scache [<session_no>]             set [OPT] [<var> [<val>]]         site <site-cmd>
    source <file>                     torrent [-O <dir>] <file|URL>...  user <user|URL> [<pass>]          wait [<jobno>]                    zcat <files>
    zmore <files>
lftp 192.168.80.128:~>
```

- centos7安装nfs服务器
- ubuntu nfs服务的搭建

ftp服务器的全称是net file system网络问卷系统，它的主要功能就是：允许网络中计算机之间能够通过TCP/IP网络共享资源

nfs服务器的搭建我们这里选择使用

- CentOS下：nfs-utils和rpcbind
- Ubuntu下：nfs-common

其搭建基本流程是：

- *nfs*工具安装
  - *Ubuntu*系列安装
  - *CentOS*系列安装
- 服务端配置
  - *Ubuntu*系列配置
  - *CentOS*系列配置
- 客户端挂载
  - *Ubuntu*系列挂载
  - *CentOS*系列挂载

0x00 nfs工具安装

不同linux发行版的安装方式不同

Debian/Ubuntu系列安装方式

```
# apt-get安装
$sudo apt-get install nfs-common
```

RedHat/Fedra/CentOs系列安装方式

```
# yum安装
$yum install nfs-utils
$yum install rpcbind
```

0x01 服务端配置

CentOS系列服务端配置如下：

- 创建共享目录；mkdir /home/test/shareDir
- 更改目录权限：chmod -R a+w shareDir/
- 更改配置文件：vim /etc/exports添加/home/test/shareDir 192.168.80.*(rw,async,no_root_squash)，每个字段的意思是：
  - /home：表示需要共享的目录
  - 192.168.80.*：指定哪些用户可以访问
    - *：所有可以ping同该主机的用户
    - 192.168.1.*：指定网段，在该网段中的用户可以挂载
    - 192.168.1.12：只有该用户能挂载

  - (ro,sync,no_root_squash)：权限
    * ro: 只读
    * rw: 读写
    * sync: 同步
    * no_root_squash：不降低root用户的权限
  - 其它信息可以通过man 5 exports进行查看
- 启动服务
  - systemctl start rpcbind
  - systemctl start nfs
- 开启rpcbind防火墙端口
  - iptables -I INPUT -p udp --dport 111 -j ACCEPT
  - iptables -I INPUT -p tcp --dport 111 -j ACCEPT
- 开发nfs防火墙端口
  - iptables -I INPUT -p udp --dport 2049 -j ACCEPT
  - iptables -I INPUT -p tcp --dport 2049 -j ACCEPT

最后我们通过showmount -e 192.168.80.128命令来查看服务端状态，确认之前创建的目录是否被共享

```
[root@localhost test]# showmount  -e  192.168.80.128
Export list for 192.168.80.128:
/home/test/shareDir 192.168.80.*
```

0x02 客户端挂载

CentOS系列客户端配置如下：

- 手动将共享目录挂载到本地指定目录下
  - mount 192.168.80.128:/home/test/shareDir /tmp/test/
- 配置开机自动挂载
  - echo "192.168.80.128:/home/test/shareDir /tmp/test/ nfs defaults 0 0" >> /etc/fstab
- 共享测试
  - 客户端创建文件：touch /tmp/test/1.txt
  - 服务端是否同步：ll /home/test/shareDir

```
[root@localhost shareDir]# ll /home/test/shareDir/
总用量 0
-rw-r--r--. 1 root root 0 3月  25 22:04 1.txt
[root@localhost shareDir]#
```

ssh服务器搭建

ssh服务器的主要功能就是：远程登录全权操作主机

ssh服务器的搭建我们这里选择使用openssh-server工具

其搭建基本流程是:

0x00 server工具安装

不同linux发行版的安装方式不同

Debian/Ubuntu系列安装方式

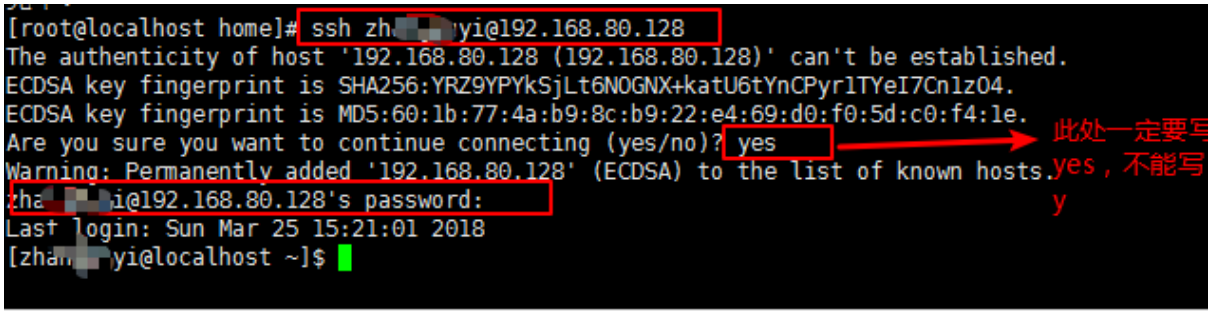
```
# apt-get安装
$sudo apt-get install openssh-server
```

RedHat/Fedra/CentOs系列安装方式

```
# yum安装
$yum install openssh-server
```

0x01 远程登陆

登录方式很简单：`ssh UserName@ServerIp`



0x02 连接操作

ssh远程登录后所能做的操作包括远程主机支持的所有本地操作

samba服务器搭建

磁盘管理

安全管理

内核管理

开发工具

常见操作

1.查看内存空间使用状态

- `cat /proc/meninfo`
- `free`(-m:显示结果以MB位单位；-g:显示结果以GB为单位)
- `vmstat -s`

2.查看当前系统挂载信息

- `mount`
- `cat /proc/mounts`
- `cat /etc/mtab`

3.查看整个目录文件 (包含目录下文件和子目录)的大小

- *du -s /path/to/dir*

4.查看挂载磁盘的分区信息

- `fdisk -l dev`查看指定磁盘的分区信息
- `cat /proc/partitions`查看所有挂载磁盘的分区信息

5.查看超级块SuperBlock的信息

- `dumpe2fs -h /deb/sdb`
- `tune2fs -l /deb/sdb`

6.启动图形界面

- `startx`启动Gnome界面
- `startkde`启动KDE界面
- `Ctrl+Alt+F7`启动图形界面

7.取消正在执行的命令

- `Ctrl+c`当执行startx时可以使用该命令退出图形界面

8.立即释放命令提示符(后台执行)

- `startx &`可以在启动完图形界面后释放命令提示符

9.翻屏

- `shift+PageUp/PageDown`

10.调用上一条命令后面的参数

- 使用`esc+.`，先`esc`键然后点号

11.在交互模式下回删字符

- ctrl+BackSpace

12.查看端口对应的服务名或者服务对应的端口

- grep ssh /etc/services，/etc/services是一个端口名称解析库

13.通过查看服务组件的生成文件查看服务的服务名

- rpm -ql telnet-server

14.关闭GUI

- init 3
- Alt+Ctrl+F1~F6
- 修改配置文件

15.启动GUI

- init 5
- startx
- 修改配置文件

其它工具

```
$export       # 显示当前shell环境变量
```

shell编程

参考文档

- Shell脚本
- man bash文档

shell可以理解为一种脚本语言，像javascript等其它脚本语言一样，只需要一个能编写代码的文本编辑器和一个能解释执行的脚本解释器就可以

shell脚本的本质是：以某种语法格式将shell命令组织起来的由shell程序解析执行的脚本文本文件

由本质可知，要想掌握shell脚本，就需要了解并掌握下列三部分内容

- **shell命令**：即ls/cd等linux命令，详细可参考shell命令
- **shell解释器**：即sh/bash/csh等shell应用程序，详细可参考shell应用程序
- **shell语法**：即数据类型/变量/控制流语句/函数等编程语法

关于shell命令和shell解释器可参考上述指定的文档，本系列主要是对shell语法进行相关讲解，将从以下方面展开介绍：

语法基础

目录

脚本结构

我们在学习每一种编程语言时，都会先学习写一个hello world的demo程序，下面我们将从这个小demo程序来窥探一下我们shell脚本的程序结构

```
#!/bin/bash

# 注释信息

echo_str="hello world"

test(){
        echo $echo_str
}

test echo_str
```

首先我们可以通过文本编辑器(在这里我们使用linux自带文本编辑神器vim)，新建一个文件demo.sh，文件扩展名sh代表shell，表明该文件是一个shell脚本文件，并不影响脚本的执行，然后将上述代码片段写入文件中，保存退出

然后使用bash -n demo.sh命令可以检测刚才脚本文件的语法是否错误，如果没有回显结果就代表脚本文件没有语法错误

关于上述脚本文件中的代码语法，这里我们简单说明下，详细说明介绍将在下述文档中一一展开

- 脚本都以#!/bin/bash开头，#称为sharp，!在unix行话里称为bang，合起来简称就是常见的shabang。#!/bin/bash 指定了shell脚本解释器bash的路径，即使用bash程序作为该脚本文件的解释器，当然也可以使用其它的解释器/bin/sh等，根据具体环境进行相应选择
- echo_str是字符串变量，通过$进行引用变量的值，
- test是自定义函数名，通过函数名 传入参数格式进行函数的调用
- echo是shell命令，相对于c中的printf
- #字符用来注释shell脚本的

最后可以使用下列两种方式执行上述脚本

- 将脚本作为bash解释器的参数执行：此时首行的#!/bin/bashshabang可以不用写
  - bash demo.sh：直接将脚本文件作为bash命令的参数
  - bash -x demo.sh：使用-x参数可以查看脚本的详细执行过程
- 将脚本作为独立的可执行文件执行：此时首行的#!/bin/bashshabang必须写，用来指定shell解释器路径；同时脚本必须可执行权限
  - chmod +x demo.sh：给脚本添加执行权限
  - ./demo.sh：执行脚本文件，在这里需要使用./demo.sh表明当前目录下脚本，因为PATH环境变量中没有当前目录，写成demo.sh系统会去/sbin、/sbin等目录下查找该脚本，无法找到该脚本文件执行，造成报错

数据类型

数据类型的本质：固定内存大小的别名

数据类型的作用：

- 确定对应变量分配的内存大小
- 确定对应变量所能支持的运算或操作

shell脚本是弱类型解释型的语言，在脚本运行时由解释器进行解释变量在什么时候是什么数据类型

在bash中，变量默认都是字符串类型，都是以字符串方式存储，所以在本章主要是介绍各数据类型变量所支持的运算或操作

虽说变量默认都是字符串类型，但是按照其使用场景可将数据类型分为以下几种类型：

- 数值型

- 字符串型
- 数组型
- 列表型

0x00 数值型

首先我们来声明定义一个数值型变量：`declare -i Var_Name`

- 虽说声明是一个数值型变量，但是存储依然是按照字符串的形式进行存储
- 该种方式声明，变量默认是本地全局变量，可以通过`local Var_Name`关键字将变量修改为局部变量，可以通过`export Var_Name`关键字将变量导出为环境变量
- 除了使用`declare -i`显式声明变量数据类型为数值型，还可以像`Var_Name=1`由解释器动态执行隐式声明该变量数据类型为数值型

数值型变量一般支持以下运算操作

- 算术运算
- 比较运算
- 数组索引

0x0000 算术运算

算术运算代码示例如下

```
#!/bin/bash

declare -i val=5    # 显式声明数值变量
num=2               # 隐式声明数值变量

# 使用[]运算符执行算术表达式$val+$num
# 使用$引用表达式执行结果
echo "val+num=$[$val+$num]"
echo "val++: $[val++]"   # 这里不需要加$，不是引用变量的值，而是修改变量的值
echo "val--: $[val--]"   # 这里不需要加$，不是引用变量的值，而是修改变量的值
echo "++val: $[++val]"   # 这里不需要加$，不是引用变量的值，而是修改变量的值
echo "--val: $[--val]"   # 这里不需要加$，不是引用变量的值，而是修改变量的值

# 使用(())运算符执行算术表达式
# 使用$引用表达式执行结果
echo "val-num=$(($val-$num))"
echo "val%num=$(($val%$num))"

# 使用let关键字执行算术表达式$val*$num
# 使用=运算符将执行结果赋值给变量
let ret=$val*$num
echo "var*num=$ret"

# 使用expr命令执行算术表达式$val/$num但是$val / $num之间需要用空格隔开
# 此时该表达式中的各个部分将作为参数传递给expr命令，最后使用``运算符引用命令的执行结果
# 使用=运算符将命令引用结果赋值给变量
ret=`expr $val / $num`
echo "val/num=$ret"

# 使用let关键字执行算术表达式+=、-=、*=、/=、%=
let val+=$num
echo "var+=num:$val"
let val-=$num
```

(continues on next page)

(continued from previous page)

```
echo "var-=num:$val"
let val*=$num
echo "val*=num:$val"
let val/=$sum          # 貌似let不支持/=运算符
echo "val/=num:$val"
let val%=$num
echo "val%=num:$val"

# 执行结果如下
# val+num=7
# val++: 5
# val--: 6
# ++val: 6
# --val: 5
# val-num=3
# val%num=1
# var*num=10
# val/num=2
# var+=num:7
# var-=num:5
# val*=num:10
# ./test.sh: line 19: let: val/=: syntax error: operand expected (error token is "/
↪=")
# val/=num:10
# val%=num:0
```

由上述示例可知：数值类型变量支持的算术运算以及对应的算术运算符如下

- 加：+、+=、++
- 减：-、-=、--
- 乘：*、*=
- 除：/
- 取余：%、%=

0x0001 比较运算

比较运算有以下几种类型

- 用于条件测试
- 用于*for*循环

用于条件测试的示例代码如下

```
#!/bin/bash

declare -i val=5    # 显式声明数值变量
num=2               # 隐式声明数值变量

# -eq: 判断val变量的值是否等于5
# []运算符用来执行条件测试表达式，其执行结果要么为真，要么为假
# []运算符和条件测试表达式之间前后有空格
if [ $val -eq 5 ]; then
        echo "the value of val variable is 5"
fi

# -ne: 判断num变量的值是否不等于5
# [[]]运算符用来执行条件测试表达式，其执行结果要么为真，要么为假
```

(continues on next page)

(continued from previous page)

```
# [[]]运算符和条件测试表达式之间前后有空格
if [[ $num -ne 5 ]];then
        echo "the value of num variable is not 5"
fi

# -le: 判断num变量的值是否小于或等于val变量的值
# test命令关键字用来执行条件测试表达式，其执行结果要么为真，要么为假
if test $num -le $val ;then
        echo "the value of num variable is lower or equal than val variable"
fi

# -ge: 判断val变量的值是否大于或等于num变量的值
# [[]]运算符用来执行条件测试表达式，其执行结果要么为真，要么为假
# [[]]运算符和条件测试表达式之间前后有空格
if [[ $val -ge $num ]];then
        echo "the value of val variable is growth or equal than num variable"
fi

# -gt: 判断val变量的值是否大于5
# []运算符用来执行条件测试表达式，其执行结果要么为真，要么为假
# []运算符和条件测试表达式之间前后有空格
if [ $val -gt 2 ];then
        echo "the value of val variable is growth than 2"
fi

# -lt: 判断num变量的值是否小于5
# [[]]运算符用来执行条件测试表达式，其执行结果要么为真，要么为假
# [[]]运算符和条件测试表达式之间前后有空格
if [[ $num -lt 5 ]];then
        echo "the value of num variable is lower than 5"
fi

# 执行结果如下
# the value of val variable is 5
# the value of num variable is not 5
# the value of num variable is lower or equal than val variable
# the value of val variable is growth or equal than num variable
# the value of val variable is growth than 2
# the value of num variable is lower than 5
```

由上述示例可知：数值类型变量用于条件测试时支持的比较运算以及对应的运算符如下

- 等于：`-eq`
- 不等于：`-ne`
- 小于等于：`-le`
- 大于等于：`-ge`
- 大于：`-gt`
- 小于：`-lt`
- 逻辑与：`&&`
- 逻辑非：`!`
- 逻辑或：`||`

用于用于for循环的示例代码如下

```
#!/bin/bash

# ==判断变量i的值是否等于1
```

(continues on next page)

(continued from previous page)

```
for ((i=1; i==1; i++));do
        echo $i
done

# !=判断变量i的值是否不等于3
for ((i=1; i!=3; i++)); do
        echo $i
done

# <=判断变量i的值是否小于等于4
for ((i=1; i<=4; i++)); do
        echo $i
done

# >=判断变量i的值是否大于等于1
for ((i=5; i>=1; i--));do
        echo $i
done

# <判断变量i的值是否小于7
# >判断变量i的值是否大于0
# &&表示逻辑与
# ||表示逻辑或
# !表示逻辑非
# 非的优先级大于与，与的优先级大于或
for ((i=1; i>0 && i<7; i++)); do
        echo $i
done
```

由上述示例可知：数值类型变量用于for循环时支持的比较运算以及对应的运算符如下

- 等于：==
- 不等于：!=
- 小于等于：<=
- 大于等于：>=
- 大于：>
- 小于：<
- 逻辑与：&&
- 逻辑非：!
- 逻辑或：||

0x0002 数组索引

数组类型变量当做数组索引可参考数组型变量一节

0x01 字符串型

首先我们来声明定义一个字符串型变量：`Var_Name="anony"`

- 在bash中，变量默认都是字符串类型，也都是以字符串方式存储，所以字符串可以不需要使用""，除非特殊声明，否则都会解释成字符串
- 该种方式声明，变量默认是本地全局变量，可以通过`local Var_Name`关键字将变量修改为局部变量，可以通过`export Var_Name`关键字将变量导出为环境变量

- 该种声明定义方式是由shell解释器动态执行隐式声明该变量数据类型为字符串型

字符串型变量一般支持以下运算操作

- 返回字符串长度：`${#Var_Name}`(长度包括空白字符)
- 字符串消除
  - `${var#*word}`：查找var中自左而右第一个被word匹配到的串，并将此串及向左的所有内容都删除；此处为非贪婪匹配
  - `${var##*word}`：查找var中自左而右最后一个被word匹配到的串，并将此串及向左的所有内容都删除；此处为贪婪匹配
  - `${var%word*}`：查找var中自右而左第一个被word匹配到的串，并将此串及向右的所有内容都删除；此处为非贪婪匹配
  - `${var%%word*}`：查找var中自右而左最后一个被word匹配到的串，并将此串及向右的所有内容都删除；此处为贪婪匹配
- 字符串提取
  - `${var:offset}`：自左向右偏移offset个字符，取余下的字串；例如：name=jerry，${name:2}结果为rry
  - `${var:offset:length}`：自左向右偏移offset个字符，取余下的length个字符长度的字串。例如：‘‘name=’hello world’ ${name:2:5}结果为llo w‘‘
- 字符串替换
  - `${var/Pattern/Replaceplacement}`：以Pattern为模式匹配var中的字串，将第一次匹配到的替换为Replaceplacement；此处为非贪婪匹配，Pattern模式可参考正则表达式
  - `${var//Pattern/Replaceplacement}`：以Pattern为模式匹配var中的字串，将全部匹配到的替换为Replaceplacement；此处为贪婪匹配，Pattern模式可参考正则表达式

代码示例如下：

```
#!/bin/bash
echo "PATH variable is $PATH"
echo "the length of PATH variable is ${#PATH}"

file_name="linux.test.md"
echo "${file_name%%.*}"
echo "${file_name%.*}"
echo "${file_name##*.}"
echo "${file_name#*.}"
echo "${file_name:0:5}"
echo "${file_name:2}"

test_str="/usr/bin:/root/bin:/usr/local/apache/bin:/usr/local/mysql:/usr/local/
↪apache/bin"
echo "${test_str/:\/usr\/local\/apache\/bin/}"    # 此处需要使用\对/进行转义，替换值为空表
示删除前面匹配到的内容
echo "${test_str//:\/usr\/local\/apache\/bin/}"   # 此处需要使用\对/进行转义，替换值为空表
示删除前面匹配到的内容

# 执行结果如下
# PATH variable is /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
# the length of PATH variable is 59
# linux
# linux.test
# md
# test.md
# linux
# nux.test.md
```

(continues on next page)

(continued from previous page)

```
# /usr/bin:/root/bin:/usr/local/mysql:/usr/local/apache/bin
# /usr/bin:/root/bin:/usr/local/mysql
```

0x02 数组型

数组是一种数据结构，也可以叫做数据序列，它是一段连续的内容空间，保存了连续的多个数据(数据类型可以不相同)，可以使用数组index索引来访问操作数组元素

根据数组index索引的不同可将数组分为

- 普通数组：数组index索引为整型数
- 关联数组：数组index索引为字符串

0x0200 普通数组

普通数组也可以称为整型索引数组，它的声明定义方式有以下几种

```
#!/bin/bash

# 使用declare -a显式声明变量数据类型为整型索引数组型
# 数组中各元素间使用空白字符分隔
# 字符串类型的元素使用引号
declare -a array1=(1 'b' 3 'a')
# 依次引用数组的第一、二、三、四个元素
# 不加下标时默认引用第一个元素
# 引用时必须加上{}，否则$array1[0]的值为1[0]
echo "the first element of array1 is ${array1[0]}"
echo "the second element of array1 is ${array1[1]}"
echo "the third element of array1 is ${array1[2]}"
echo "the fourth element of array1 is ${array1[3]}"
# 查看数组所有元素
echo "all elements of array1 is ${array1[*]}"
echo "all elements of array1 is ${array1[@]}"


# 由解释器动态解释变量数据类型为整型索引数组型
# 如果数组中各元素间使用逗号，则它们将作为一个整体，也就是数组索引0的值
array2=(1,'b',3,'a')
echo "the first element of array2 is ${array2[0]}"


# 由解释器动态解释变量数据类型为整型索引数组型
# 数组元素使用自定义下标赋值
# 以下数组定义中，第一个元素是1，第二个元素是'b'，第3个元素为空，第4个元素为'a'
array3=(1 'b' [3]='a')
# 依次引用数组的第一、二、三、四个元素
# 不加下标时默认引用第一个元素
echo "the first element of array3 is ${array3[0]}"
echo "the second element of array3 is ${array3[1]}"
echo "the third element of array3 is ${array3[2]}"
echo "the fourth element of array3 is ${array3[3]}"
# 查看数组中所有有效元素(不为空)的整型索引号
echo "the index of effective element is ${!array3[*]}"
echo "the index of effective element is ${!array3[@]}"
# 查看数组中的有效元素个数(只统计值不为空的元素)
echo "the num of array3 is ${#array3[*]}"
echo "the num of array3 is ${#array3[@]}"
```

(continues on next page)

(continued from previous page)

```
# 由解释器动态解释变量数据类型为整型索引数组型
# 数组中每个元素被逐渐赋值
array4[0]=1
array4[1]='bc'
array4[2]=3
array4[3]='a'
# 依次引用数组的第一、二、三、四个元素
# 不加下标时默认引用第一个元素
echo "the first element of array4 is ${array4[0]}"
echo "the second element of array4 is ${array4[1]}"
echo "the third element of array4 is ${array4[2]}"
echo "the fourth element of array4 is ${array4[3]}"
# 查看第二个元素的字符长度
echo "the length of second element is ${#array4[1]}"


# 执行结果如下
# the first element of array1 is 1
# the second element of array1 is b
# the third element of array1 is 3
# the fourth element of array1 is a
# all elements of array1 is 1 b 3 a
# all elements of array1 is 1 b 3 a
# the first element of array2 is 1,b,3,a
# the first element of array3 is 1
# the second element of array3 is b
# the third element of array3 is
# the fourth element of array3 is a
# the index of effective element is 0 1 3
# the index of effective element is 0 1 3
# the num of array3 is 3
# the num of array3 is 3
# the first element of array4 is 1
# the second element of array4 is bc
# the third element of array4 is 3
# the fourth element of array4 is a
# the length of second element is 2
```

另外普通数组还支持以下运算操作

- 返回数组长度(即有效元素的个数，不包括空元素)
  - `${#Array_Name[*]}`
  - `${#Array_Name[@]}`
- 数组元素消除，该操作不会修改原数组元素，操作执行结果用数组来接收
  - `Array_Name1=${Array_Name[*]#*word}`：功能同下
  - `Array_Name1=${Array_Name[*]##*word}`：自左而右查找`Array_Name`数组中所有被匹配到的`word`匹配到的元素，并将所有匹配到的元素删除(并不会删除原数组中的元素)，最后返回剩余的数组元素
  - `Array_Name1=${Array_Name[*]%word*}`：功能同下
  - `Array_Name1=${Array_Name[*]%%word*}`：自右而左查找`Array_Name`数组中所有被匹配到的`word`匹配到的元素，并将所有匹配到的元素删除(并不会删除原数组中的元素)，最后返回剩余的数组元素
- 数组元素提取，该操作不会修改原数组元素，操作执行结果用数组来接收
  - `Array_Name1=${Array_Name[*]:offset}`：返回`Array_Name`数组中索引为`offset`的数组元素以及后面所有元素；其中`offset`为整型数

  - `Array_Name1=${Array_Name[*]:offset:length}`：返回`Array_Name`数组中索引为`offset`的数值元素以及后面`length-1`个元素；其中`offset`和`length`都为整型数

- 数组元素替换，该操作不会修改原数组元素，操作执行结果用数组来接收
  - `Array_Name1=${Array_Name[*]/Pattern/Replaceplacement}`：功能同下
  - `Array_Name1=${Array_Name[*]//Pattern/Replaceplacement}`：以`Pattern`为模式匹配`Array_Name`数组中的元素，将全部匹配到的替换为`Replaceplacement`(不会修改原数组中的元素)，并返回全部数组元素；`Pattern`模式可参考正则表达式

代码示例如下

```
#!/bin/bash

array_test=(/usr/bin /root/bin /usr/apache/bin /usr/mysql /usr/apache/bin)

# 返回数组长度(即有效元素的个数，不包括空元素)
echo "the length of array_test is ${#array_test[*]}"
echo "the length of array_test is ${#array_test[@]}"

# 数组元素消除，该操作不会修改原数组元素，操作执行结果用数组来接收
array_test1=${array_test[*]#*/usr/apache/bin}
echo "array_test:${array_test[*]}"
echo "array_test1:${array_test1[@]}"
array_test2=${array_test[*]##*/usr/apache/bin}
echo "array_test:${array_test[*]}"
echo "array_test2:${array_test2[@]}"
array_test3=${array_test[*]%/usr/apache/bin*}
echo "array_test:${array_test[*]}"
echo "array_test3:${array_test3[@]}"
array_test4=${array_test[*]%%/usr/apache/bin*}
echo "array_test:${array_test[*]}"
echo "array_test4:${array_test4[@]}"

# 数组元素提取，该操作不会修改原数组元素，操作执行结果用数组来接收
array_test5=${array_test[*]:2}
echo "array_test:${array_test[*]}"
echo "array_test5:${array_test5[@]}"
array_test6=${array_test[*]:2:2}
echo "array_test:${array_test[*]}"
echo "array_test6:${array_test6[@]}"

# 数组元素替换，该操作不会修改原数组元素，操作执行结果用数组来接收
array_test7=${array_test[*]/\/usr\/apache\/bin/}    # 需要用\对/进行转义，替换值为空表示删除前面匹配到的
echo "array_test:${array_test[*]}"
echo "array_test7:${array_test7[@]}"
array_test8=${array_test[*]//\/usr\/apache\/bin/}   # 需要用\对/进行转义，替换值为空表示删除前面匹配到的
echo "array_test:${array_test[*]}"
echo "array_test8:${array_test8[@]}"

# 执行结果如下
# the length of array_test is 5
# the length of array_test is 5
# array_test:/usr/bin /root/bin /usr/apache/bin /usr/mysql /usr/apache/bin
# array_test1:/usr/bin /root/bin /usr/mysql
# array_test:/usr/bin /root/bin /usr/apache/bin /usr/mysql /usr/apache/bin
# array_test2:/usr/bin /root/bin /usr/mysql
# array_test:/usr/bin /root/bin /usr/apache/bin /usr/mysql /usr/apache/bin
# array_test3:/usr/bin /root/bin /usr/mysql
# array_test:/usr/bin /root/bin /usr/apache/bin /usr/mysql /usr/apache/bin
# array_test4:/usr/bin /root/bin /usr/mysql
```

(continues on next page)

(continued from previous page)

```
# array_test:/usr/bin /root/bin /usr/apache/bin /usr/mysql /usr/apache/bin
# array_test5:/usr/apache/bin /usr/mysql /usr/apache/bin
# array_test:/usr/bin /root/bin /usr/apache/bin /usr/mysql /usr/apache/bin
# varray_test6:/usr/apache/bin /usr/mysql
# array_test:/usr/bin /root/bin /usr/apache/bin /usr/mysql /usr/apache/bin
# array_test7:/usr/bin /root/bin /usr/mysql
# array_test:/usr/bin /root/bin /usr/apache/bin /usr/mysql /usr/apache/bin
# array_test8:/usr/bin /root/bin /usr/mysql
```

同时普通数组也可用于for循环遍历

代码示例如下

```
#!/bin/bash

# 获取家目录下文件列表，转换成普通数组
array_test=(`ls ~`)
echo ${array_test[@]}
echo "-----------------"

# 以数组元素值的方式直接遍历数组
for i in ${array_test[*]};do
        echo $i
done
echo "-----------------"

# 以数组index索引的方式遍历数组
for i in ${!array_test[*]};do
        echo ${array_test[$i]}
done
echo "-----------------"

# 以数组元素个数的方式遍历数组
for ((i=0;i<${#array_test[*]};i++));do
        echo ${array_test[$i]}
done

# 执行结果如下
# anaconda-ks.cfg demo.sh test1.sh test.sh
# -----------------
# anaconda-ks.cfg
# emo.sh
# est1.sh
# test.sh
# -----------------
# anaconda-ks.cfg
# demo.sh
# test1.sh
# test.sh
# -----------------
# anaconda-ks.cfg
# demo.sh
# test1.sh
# test.sh
```

0x0201 关联数组

关联数组也可以称为字符索引数组，它的声明定义方式有以下几种

```
#!/bin/bash

# 声明定义字符索引数组时必须使用declare -A
# 数组中各元素间使用空白字符分隔
declare -A array1=([name1]=jack [name2]=anony)
# 依次引用name1和name2对应的值
echo "the value of name1 element is ${array1[name1]}"
echo "the value of name2 element is ${array1[name2]}"


# 声明定义字符索引数组时必须使用declare -A
# 如果数组中各元素间使用逗号，则它们将作为一个整体
declare -A array2=([name1]=jack,[name2]=anony)
echo "the value of name1 element is ${array2[name1]}"
# 查看name1对应值的字符长度
echo "the length of name1 element is ${#array2[name1]}"


# 声明定义字符索引数组时必须使用declare -A
declare -A array3=([name1]=jack [name2]=anony)
echo "the value of name1 element is ${array3[name1]}"
echo "the value of name2 element is ${array3[name2]}"
# 通过字符索引进行赋值
array3[name3]=zhangsan
echo "the value of name3 element is ${array3[name3]}"
# 通过字符索引进行赋值
array3[name5]=lisi
# 查看数组所有元素
echo "the all effective element is ${array3[*]}"
echo "the all effective element is ${array3[@]}"
# 查看数组中所有有效元素(不为空)的字符索引号，默认是对应值的排列顺序
echo "the index of all effective element is ${!array3[*]}"
echo "the index of all effective element is ${!array3[@]}"
# 查看数组中的有效元素个数(只统计值不为空的元素)
echo "the length of array is ${#array3[*]}"
echo "the length of array is ${#array3[@]}"

# 执行结果如下
# the value of name1 element is jack
# the value of name2 element is anony
# the value of name1 element is jack,[name2]=anony
# the length of name1 element is 18
# the value of name1 element is jack
# the value of name2 element is anony
# the value of name3 element is zhangsan
# the all effective element is zhangsan anony jack lisi
# the all effective element is zhangsan anony jack lisi
# the index of all effective element is name3 name2 name1 name5
# the index of all effective element is name3 name2 name1 name5
# the length of array is 4
# the length of array is 4
```

和普通数组一样，关联数组也支持以下运算操作

- 返回数组长度(即有效元素的个数，不包括空元素)
  - `${#Array_Name[*]}`
  - `${#Array_Name[@]}`
- 数组元素消除，该操作不会修改原数组元素，操作执行结果用数组来接收
  - `declare -A Array_Name1=${Array_Name[*]#*word}`: 功能同下
  - `declare -A Array_Name1=${Array_Name[*]##*word}`: 自左而右查

找Array_Name数组中所有被匹配到的word匹配到的元素，并将所有匹配到的元素删除(并不会删除原数组中的元素)，最后返回剩余的数组元素

  - declare -A Array_Name1=${Array_Name[*]%word*}：功能同下
  - declare -A Array_Name1=${Array_Name[*]%%word*}：自右而左查找Array_Name数组中所有被匹配到的word匹配到的元素，并将所有匹配到的元素删除(并不会删除原数组中的元素)，最后返回剩余的数组元素

- 数组元素提取，该操作不会修改原数组元素，操作执行结果用数组来接收
  - declare -A Array_Name1=${Array_Name[*]:offset}：返回Array_Name数组中索引为offset的数组元素以及后面所有元素；其中offset为整型数
  - declare -A Array_Name1=${Array_Name[*]:offset:length}：返回Array_Name数组中索引为offset的数值元素以及后面length-1个元素；其中offset和length都为整型数
- 数组元素替换，该操作不会修改原数组元素，操作执行结果用数组来接收
  - declare -A Array_Name1=${Array_Name[*]/Pattern/Replaceplacement}：功能同下
  - declare -A Array_Name1=${Array_Name[*]//Pattern/Replaceplacement}：以Pattern为模式匹配Array_Name数组中的元素，将全部匹配到的替换为Replaceplacement(不会修改原数组中的元素)，并返回全部数组元素；Pattern模式可参考正则表达式

代码示例如下

```
#!/bin/bash

declare -A array_test=([ele1]=/usr/bin [ele2]=/root/bin [ele3]=/usr/apache/bin␣
↪[ele4]=/usr/mysql [ele5]=/usr/apache/bin)

# 返回数组长度(即有效元素的个数，不包括空元素)
echo "the length of array_test is ${#array_test[*]}"
echo "the length of array_test is ${#array_test[@]}"

# 数组元素消除，该操作不会修改原数组元素，操作执行结果用数组来接收
declare -A array_test1=${array_test[*]#*/usr/apache/bin}
echo "array_test:${array_test[*]}"
echo "array_test1:${array_test1[@]}"
declare -A array_test2=${array_test[*]##*/usr/apache/bin}
echo "array_test:${array_test[*]}"
echo "array_test2:${array_test2[@]}"
declare -A array_test3=${array_test[*]%/usr/apache/bin*}
echo "array_test:${array_test[*]}"
echo "array_test3:${array_test3[@]}"
declare -A array_test4=${array_test[*]%%/usr/apache/bin*}
echo "array_test:${array_test[*]}"
echo "array_test4:${array_test4[@]}"

# 数组元素提取，该操作不会修改原数组元素，操作执行结果用数组来接收
declare -A array_test5=${array_test[*]:2}
echo "array_test:${array_test[*]}"
echo "array_test5:${array_test5[@]}"
declare -A array_test6=${array_test[*]:2:2}
echo "array_test:${array_test[*]}"
echo "array_test6:${array_test6[@]}"

# 数组元素替换，该操作不会修改原数组元素，操作执行结果用数组来接收
declare -A array_test7=${array_test[*]/\/usr\/apache\/bin/}
echo "array_test:${array_test[*]}"
echo "array_test7:${array_test7[@]}"
```

(continues on next page)

(continued from previous page)

```
declare -A array_test8=${array_test[*]//\/usr\/apache\/bin/}
echo "array_test:${array_test[*]}"
echo "array_test8:${array_test8[@]}"

# 执行结果如下
# the length of array_test is 5
# the length of array_test is 5
# array_test:/usr/mysql /usr/apache/bin /usr/bin /root/bin /usr/apache/bin
# array_test1:/usr/mysql  /usr/bin /root/bin
# array_test:/usr/mysql /usr/apache/bin /usr/bin /root/bin /usr/apache/bin
# array_test2:/usr/mysql  /usr/bin /root/bin
# array_test:/usr/mysql /usr/apache/bin /usr/bin /root/bin /usr/apache/bin
# array_test3:/usr/mysql  /usr/bin /root/bin
# array_test:/usr/mysql /usr/apache/bin /usr/bin /root/bin /usr/apache/bin
# array_test4:/usr/mysql  /usr/bin /root/bin
# array_test:/usr/mysql /usr/apache/bin /usr/bin /root/bin /usr/apache/bin
# array_test5:/usr/apache/bin /usr/bin /root/bin /usr/apache/bin
# array_test:/usr/mysql /usr/apache/bin /usr/bin /root/bin /usr/apache/bin
# array_test6:/usr/apache/bin /usr/bin
# array_test:/usr/mysql /usr/apache/bin /usr/bin /root/bin /usr/apache/bin
# array_test7:/usr/mysql  /usr/bin /root/bin
# array_test:/usr/mysql /usr/apache/bin /usr/bin /root/bin /usr/apache/bin
# array_test8:/usr/mysql  /usr/bin /root/bin
```

关联数组和普通数组一样，也可用于for循环遍历

先创建`test.log`文件，内容如下

```
#cat ~/test.log
portmapper
portmapper
portmapper
portmapper
portmapper
portmapper
status
status
mountd
mountd
mountd
mountd
mountd
mountd
nfs
nfs
nfs_acl
nfs
nfs
nfs_acl
nlockmgr
nlockmgr
nlockmgr
nlockmgr
nlockmgr
nlockmgr
```

代码示例如下：统计文件中重复行的次数

```
#!/bin/bash

declare -A array_test
```

(continues on next page)

(continued from previous page)

```
for i in `cat ~/test.log`;do
        let ++array_test[$i]  # 修改数组元素值
done

for j in ${!array_test[*]};do
        printf "%-15s %3s\n" $j :${array_test[$j]}
done

# 执行结果如下
# status            :2
# nfs               :4
# portmapper        :6
# nlockmgr          :6
# nfs_acl           :2
# mountd            :6
```

0x03 列表型

列表型变量常用来for循环遍历，但是一般是在for循环中直接使用，当然也可以通过变量进行引用

代码示例如下

```
#!/bin/bash

# 生成数字列表：使用{}运算符
for i in {1..4};do
        echo $i
done
echo "--------------------"

# 生成数字列表：使用seq命令
for i in `seq 1 2 7`;do
        echo $i
done
echo "--------------------"

# 生成文件列表：直接给出列表
for fileName in /etc/init.d/functions /etc/rc.d/rc.sysinit /etc/fstab;do
        echo $fileName
done
echo "--------------------"

# 生成文件列表：使用文件名通配机制生成列表
dirName=/etc/rc.d
for fileName in $dirName/*.d;do
        echo $fileName
done
echo "--------------------"

# 生成文件列表：使用``运算符引用相关命令的执行结果
for fileName in `ls ~`;do
        echo $fileName
done

# 执行结果如下
# 1
# 2
# 3
#4
```

(continues on next page)

(continued from previous page)

```
# --------------------
# 1
# 3
# 5
# 7
# --------------------
# /etc/init.d/functions
# /etc/rc.d/rc.sysinit
# /etc/fstab
# --------------------
# /etc/rc.d/init.d
# /etc/rc.d/rc0.d
# /etc/rc.d/rc1.d
# /etc/rc.d/rc2.d
# /etc/rc.d/rc3.d
# /etc/rc.d/rc4.d
# /etc/rc.d/rc5.d
# /etc/rc.d/rc6.d
# --------------------
# anaconda-ks.cfg
# demo.sh
# test1.sh
# test.log
# test.sh
```

变量

变量是一种逻辑概念，变量有三要素(也可称为三种属性)

- 数据类型：变量存储数据的类型；用来确定该变量存储数据的内存大小以及存储数据所能支持的运算操作(解释器执行解释)
- 变量类型：变量名的类型；用来确定该变量的作用域以及生命周期(关键字修饰决定)
- 变量名：访问变量存储的数据；用来访问一段可读可写的连续内存空间(自定义命名)

其中数据类型属性将在数据类型一章内容介绍

在本章内容中主要介绍

- 变量名
- 变量类型
  - 本地变量
  - 局部变量
  - 环境变量
  - 位置变量
  - 特殊变量
  - 变量属性
  - 变量赋值

0x00 变量名

变量是通过变量名进行声明、定义、赋值和引用；变量存在于内存中，对于shell变量而言，设置或修改变量属性以及变量值时，不需要带$，只有引用变量的值时才使用$

变量名的本质是：一段可读可写的连续内存空间的别名

通过对变量名的引用就可以读写访问连续的内存空间

变量名的命名须遵循如下规则：

- 命名只能使用英文字母，数字和下划线，首个字符不能以数字开头
- 中间不能有空格，可以使用下划线_
- 不能使用标点符号
- 不能使用bash内嵌的关键字(可用help命令查看保留关键字)
- 不能使用shell命令

0x01 变量类型

shell脚本是弱类型解释型的语言，变量类型由不同关键字声明决定；根据变量类型可将变量分为：(变量类型即变量名的类型，它决定变量的作用域以及定义引用的方式)

- 本地变量
- 局部变量
- 环境变量
- 位置变量
- 特殊变量
- 变量属性

0x0100 本地变量

本地变量可以理解为全局变量，它的作用域为：只对当前shell进程有效，对其子shell以及其他shell都无效

该类型变量的声明定义方式为：`[set]Var_Name=Value`

- set关键字可以省略
- 等号左右没有空格；如果有空格就是进行比较运算符的比较运算
- 该变量可以声明定义在脚本的任何地方
- 变量Var_Name可以是任意数据类型

该类型变量的引用方式(获取变量的值)为：`$Var_Name`或`${Var_Name}`

- 可以在脚本的任意地方引用

该类型变量的赋值方式(修改变量的值)为：`Var_Name=Value`

- 在脚本中任意地方的赋值都会覆盖之前的变量值

该类型变量的撤销释放方式为：`unset Var_Name`

- 变量名前不加前缀$
- 撤销该变量后，引用该变量就会为空

需要注意的是：

- 如果使用readonly关键字修饰变量Var_Name，即`readonly Var_Name[=Value]`，此时将无法修改变量值也无法unset变量
- 不接收任何参数的set或者declare关键字命令，将会输出当前所有有效的本地变量、局部变量和环境变量

示例程序如下

```
#!/bin/bash

test_str="hello world"
readonly ro_str="test"
test_one(){
        echo "test_str in test_one is $test_str"
        test_str="happy"
        test_name="anony"
        unset test_set    # 撤销变量test_set，之后引用该变量就会为空
        echo "test_set in test_one is $test_set"
        ro_str="tset"     # 该变量被readonly修饰，不能修改其变量值，将会出现语法错误，直接退出函数，不执行下列命令
        echo "ro_str"     # 上述直接退出函数，该命令不会执行
}

test_two()
{
        echo "test_str in test_two is $test_str"
        echo "test_name in test_two is $test_name"
        echo "test_set in test_two is $test_set"

}

test_one
test_two

# 执行结果如下
# test_str in test_one is hello world
# test_set in test_one is               # echo显示为空
# ./demo.sh: line 9: ro_str: readonly variable
# test_str in test_two is happy
# test_name in test_two is anony
# test_set in test_two is               # echo显示为空
```

0x0101 局部变量

局部变量的作用域为：只对变量声明定义所在函数内有效

该类型变量的声明定义方式为：`loca Var_Name=Value`

- `local`关键字不能省略，否则就是本地全局变量
- 等号左右没有空格；如果有空格就是进行比较运算符的比较运算
- 该变量只能声明定义在函数体内，否则会语法报错
- 变量Var_Name可以是任意数据类型

该类型变量的引用方式(获取变量的值)为：`$Var_Name`或`${Var_Name}`

- 只能在声明定义的函数体内引用，其它地方引用将为空

该类型变量的赋值方式(修改变量的值)为：`Var_Name=Value`

该类型变量的撤销释放方式为：`unset Var_Name`

- 变量名前不加前缀`$`
- 撤销该变量后，引用该变量就会为空

需要注意的是：

- 如果使用`readonly`关键字修饰变量`Var_Name`，即`readonly Var_Name[=Value]`，此时将无法修改变量值也无法unset变量

- 不接收任何参数的set或者declare关键字命令，将会输出当前所有有效的本地变量、局部变量和环境变量

示例程序如下

```
#!/bin/bash

test_str="anony"
test_one(){
        local test_str="happy"    # 局部变量test_str会覆盖全局变量test_str
        local test_local="test"
        echo "test_str in test_one is $test_str"
        echo "test_local in test_one is $test_local"
        unset test_str           # 只会撤销局部变量test_str，不会撤销全局变量test_str
}

test_two()
{
        echo "test_str in test_two is $test_str"       # unset没有撤销全局变量test_str
        echo "test_local in test_two is $test_local"   # test_local是定义在test_one函
数中的局部变量，该处引用将会为空

}

test_one
test_two

# 执行结果如下
# test_str in test_one is happy
# test_local in test_one is test
# test_str in test_two is anony
# test_local in test_two is
```

0x0102 环境变量

环境变量可以用来

- 定义bash的工作特性
- 保存当前会话的属性信息

关于环境变量的生命周期和作用域可以参考：bash环境配置

shell环境变量有两种来源

- 系统环境变量
  - 该环境变量已经由bash定义初始化，不用重新声明定义，只要引用就可以
    - 使用env、export、set、declare或printenv可以查看当前用户的环境变量(包括系统环境变量和自定义环境变量)，以下列出部分bash默认系统环境变量(set和declare可以查看所有环境变量，其它三个命令只能查看部分环境变量)
      - $BASH：bash二进制程序文件的路径
      - $BASH_SUBSHELL：子shell的层次说明，说明用户在哪一个层次中
      - $BASH_VERSION：bash的版本
      - $EDITOR：指定默认编辑器
      - $EUID：有效的用户ID
      - $UID：当前用户的ID号
      - $USER：当前用户名

            - $PATH：自动搜索路径
            - $LANG：系统使用语系
            - LOGNAME：当前登录的用户
            - $FUNCNAME：当前函数的名称，在函数中引用想判断自己是什么函数
            - $GROUPS：当前用户所属的组
            - $HOME：当前用户的家目录
            - $HOSTTYPE：主机架构类型，用来识别系统硬件平台
            - $MACHTYPE：平台类型，系统平台依赖的编译平台
            - $OSTYPE：OS系统类型
            - $IFS：输入数据时的默认字段分隔符，默认是空白符(空格、制表符、换行符)
            - $OLDPWD：上次使用的目录
            - $PWD：当前目录
            - $PPID：父进程
            - $PS1：主提示符，即bash命令窗口提示符
            - $PS2：第二提示符，主要用于补充完全命令输入时的提示符
            - $PS3：第三提示符，用于select命令中
            - $PS4：第四提示符，当使用-X选项调用脚本时，显示的提示符，默认为+号
            - $SECONDS：当前脚本已经运行的时长，单位为秒
            - $SHLVL：shell的级别，bash被嵌入的深度
        * 系统环境变量常用大写字母表示
    - 系统环境变量作用域
        * 执行脚本前，原始系统环境变量对当前用户所有shell进程(包含不同终端bash进程以及其子shell进程)都有效
        * 执行脚本时，系统环境变量对当前shell进程以及子shell进程都有效
        * 执行脚本后
            - 如果使用source命令执行脚本，修改后的系统环境变量会覆盖之前的系统环境变量，但是修改后的变量值只对当前终端bash进程以及其子shell进程才有效；原始变量值依然对当前用户所有shell进程(包含不同终端bash进程以及其子shell进程)都有效
            - 如果使用./demo.sh和bash demo.sh执行脚本，修改后的系统环境变量不会覆盖之前的系统环境变量，即所以系统环境变量依然保持原值，依然对当前用户所有shell进程(包含不同终端bash进程以及其子shell进程)都有效
- 自定义环境变量
    - 该环境变量是使用export命令将全局变量或局部变量导出成环境变量，需要手动声明定义
        * 方式一：export Var_Name=Value
        * 方式二：Var_Name=Value、export Var_Name
        * 自定义环境变量名尽量避免与系统环境变量名冲突；等号左右没有空格；如果有空格就是进行比较运算符的比较运算
        * 变量Var_Name可以是全局变量或局部变量，也可以是任意数据类型
    - 自定义环境变量作用域
        * 执行脚本时，自定义环境变量才被声明定义，同时继承全局变量或局部变量的作用域
        * 执行脚本后

- 如果使用./demo.sh和bash demo.sh执行脚本，自定义环境变量不会导出成系统环境变量，即脚本执行完胡，该类环境变量会自动撤销
- 如果使用source demo.sh执行脚本，只有全局环境变量才能导出成bash环境变量，局部环境变量会自动被撤销；但是导出后的全局环境变量只对当前终端bash进程以及其子shell进程才有效

不管是系统环境变量还是自定义环境变量都可以通过以下方式进行引用(获取环境变量的值)：$Var_Name或${Var_Name}

- 在环境变量的作用域之内引用
- 变量名Var_Name可以是系统环境变量名，又可以是自定义环境变量名

不管是系统环境变量还是自定义环境变量都可以通过以下方式进行赋值(修改环境变量的值)：对当前shell进程来说通过该方式赋值修改的环境变量继承之前的作用域

- 方式一：export Var_Name=Value
- 方式二：Var_Name=Value、export Var_Name

不管是系统环境变量还是自定义环境变量都可以通过下列方式进行撤销释放：unset Var_Name

- 变量名前不加前缀$
- 撤销该变量后，引用该变量就会为空

需要注意的是：

- 如果使用readonly关键字修饰变量Var_Name，即readonly Var_Name[=Value]，此时将无法修改变量值也无法unset变量
- 不接收任何参数的set或者declare关键字命令，将会输出当前所有有效的本地变量、局部变量和环境变量

示例程序如下

```
#!/bin/bash

test_one(){
        PATH=./:$PATH             # 修改系统环境变量的值
        export PATH               # 导出系统环境变量使其生效
        export MYNAME="anony"     # 将全局变量导出成环境变量
        local MYSEX="man"         # 定义局部变量
        export MYSEX              # 将局部变量导出成环境变量
        export MYBLOG="blog"
        export MYAGE="22"
        echo "PATH in test_one is $PATH"       # 上述所有定义的环境变量都有效
        echo "MYNAME in test_one is $MYNAME"
        echo "MYSEX in test_one is $MYSEX"
        echo "MYBLOG in test_one is $MYBLOG"
        echo "MYAGE in test_one is $MYAGE"
        unset MYBLOG              # 撤销全局变量导出成的环境变量
        readonly MYAGE            # 将全局变量导出成的环境变量修改为只读变量
        MYAGE="23"                # 对只读变量进行赋值修改会造成语法错误
}

test_two()
{
        echo "PATH in test_two is $PATH"          # 系统变量的作用域
        echo "MYNAME in test_two is $MYNAME"      # 全局环境变量的作用域
        echo "MYSEX in test_two is $MYSEX"        # 局部环境变量的作用域
        echo "MYBLOG in test_two is $MYBLOG"      # 全局环境变量已经撤销
        echo "MYAGE in test_two is $MYAGE"        # 全局环境变量只读
}

test_one
```

(continues on next page)

(continued from previous page)

```
test_two


# 执行结果如下
# PATH in test_one is ./:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/
↪bin
# MYNAME in test_one is anony
# MYSEX in test_one is man
# MYBLOG in test_one is blog
# MYAGE in test_one is 22
# ./demo.sh: line 20: MYAGE: readonly variable
# PATH in test_two is ./:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/
↪bin
# MYNAME in test_two is anony
# MYSEX in test_two is
# MYBLOG in test_two is
# MYAGE in test_two is 22
```

0x0103 位置变量

位置变量无需声明定义，直接引用即可；该变量也不能被赋值修改，甚至不能被unset撤销

位置变量是用来实现

- 在函数体外直接引用脚本的传入参数，它引用方式如下
  - `$0`：引用脚本名
  - `$1`：引用脚本的第1个传入参数
  - `$n`：引用脚本的第n个传入参数
- 在函数体内直接引用函数的传入参数，它引用方式如下
  - `$0`：引用脚本名
  - `$1`：引用函数的第1个传入参数
  - `$n`：引用函数的第n个传入参数

示例程序如下

```
#!/bin/bash
echo "script name is $0"

echo "the script first arg is $1"  # 引用脚本的第一个传入参数
test(){
        echo "script name is $0"
        echo "the func first arg in test is $1" # 引用函数的第一个传入参数，不是脚本的第一
个参数
}
test 26

# 执行结果如下：./test.sh 12
# script name is ./test.sh
# the script first arg is 12
# script name is ./test.sh
# the func first arg in test is 26
```

0x0104 特殊变量

特殊变量也无需声明定义，直接引用即可；该变量也不能被赋值修改，甚至不能被unset撤销

特殊变量的引用方式如下

- $?：引用上一条命令的执行状态返回值，状态用数字表示：0-255
  - 0：表示成功
  - 1-255：表示失败；需要注意的是1/2/127/255是系统预留的，自己写脚本时要避开与这些值重复
- $$：引用当前shell的PID。除了执行bash命令和shell脚本时，$$不会继承父shell的值，其他类型的子shell都继承
- $BASHPID：引用当前shell的PID，这和$$是不同的，因为每个shell的$BASHPID是独立的，而$$有时候会继承父shell的值
- $!：引用最近一次执行的后台进程PID，即运行于后台的最后一个作业的PID
- $#：引用所有位置参数的个数
- $*：引用所有位置参数的整体，即所有参数被当做一个字符串
- $@：引用所有单个位置参数，即每个参数都是一个独立的字符串
- $_：引用上一条命令的最后一个参数的值
- $-：引用传递给脚本的标记

示例程序如下

```
#!/bin/bash

echo '$# is:'$#
echo '$* is:'$*
echo '$@ is:'$@
echo '$! is:'$!
echo '$$ is:'$$
echo '$BASHPID is:'$BASHPID
echo '$? is:'$?
test(){
        echo '$# in func is:'$#
        echo '$* in func is:'$*
        echo '$@ in func is:'$@
        echo '$! in func is:'$!
        echo '$$ in func is:'$$
        echo '$BASHPID in func is:'$BASHPID
        echo '$? in func is:'$?
}
test 26 23 47

# 执行结果如下：[root@localhost ~]# ./test.sh 1 3 4 5 6 7
# $# is:6
# $* is:1 3 4 5 6 7
# $@ is:1 3 4 5 6 7
# $! is:
# $$ is:4002
# $BASHPID is:4002
# $? is:0
# $# in func is:3
# $* in func is:26 23 47
# $@ in func is:26 23 47
# $! in func is:
# $$ in func is:4002
# $BASHPID in func is:4002
# $? in func is:0
```

0x0105 变量属性

此处的变量属性是指数据类型和变量类型，这两个属性可以通过相关命令关键字进行修改，例如：

`Var_Name=Value`语句中声明定义的变量`Var_Name`默认的数据类型是字符串类型，变量类型是本地全局变量

- `local Var_Name`声明该变量为局部变量
- `export Var_Name`声明该变量为环境变量
- `declare -x Var_Name`声明该变量为环境变量
- `declare +x Var_Name`取消该变量的环境变量属性
- `declare -i Var_Name`声明该变量为整型变量
- `declare +i Var_Name`取消该变量的整型变量属性
- `declare -p Var_Name`显式指定变量被声明的类型
- `declare -r Var_Name`声明该变量为只读变量，不能撤销，不能修改，相当于readonly，只有当前进程终止才消失
- `declare +r Var_Name`取消该变量的只读变量属性

可以使用`man declare`查看`declare`命令的详细使用方法

0x0106 变量赋值

除了上述介绍的`Var_Name=Value`赋值方式，还有以下变量赋值的方式，以下赋值方式常用来给变量赋默认值

- `${var:-default}`：如果var没有声明或者声明了为空，则返回default代表的值；如果var声明了不为空，则返回var代表的值
- `${var-default}`：如果var没有声明，则返回default代表的值；如果var声明了但是为空，则返回null；如果var声明了不为空，则返回var代表的值
- `${var:+default}`：如果var没有声明或者声明了为空，不做任何操作，返回空；如果var声明了不为空，则返回default代表的值
- `${var:=default}`：如果var没有声明或者声明了为空，则返回default代表的值，并将default的值赋值给var；如果var声明了不为空，则返回var代表的值
- `${var:?default}`：如果var没有声明或者声明了为空，则以default为错误信息返回；如果var声明了不为空，则返回var代表的值

操作符

shell中常用的操作符有

- 引用操作符
- 算术操作符
- 条件测试操作符
  - 整数条件测试
  - 字符条件测试
  - 文件条件测试
- 逻辑操作符
- 括号操作符

0x00 引用操作符

引用操作符如下

- 变量引用：引用变量值，两者等效
  - `$variable`
  - `${variable}`
- 命令引用：引用命令的执行结果
  - `` `command` ``
  - `$(command)`
- 字符引用：引用字符串值
  - `''`：强引用，该操作符的优先级大于`$`，即不会进行变量替换，直接引用显示全部字符
  - `""`：弱引用，该操作符的优先级小于`$`，即先进行变量替换，然后再引用显示全部字符

0x01 算术操作符

组成算术表达式的操作符有

- 加：`+`、`+=`、`++`
- 减：`-`、`-=`、`--`
- 乘：`*`、`*=`
- 除：`/`
- 取余：`%`、`%=`

执行算术表达式的操作符有

- `$[算术表达式]`
- `$((算术表达式))`

0x02 条件测试操作符

条件测试有以下几种情况

- 整数条件测试
- 字符条件测试
- 文件条件测试

0x0200 整数条件测试

组成整数条件测试表达式的操作符有

- `-eq`：等于
- `-ne`：不等于
- `-le`：小于等于
- `-ge`：大于等于
- `-lt`：小于
- `-gt`:大于

执行整数条件测试表达式的操作符有

- `[ 整数条件测试表达式 ]`：前后有空格
- `[[ 整数条件测试表达式 ]]`：前后有空格

0x0201 字符条件测试

组成字符条件测试表达式的操作符有

- `>`：大于
- `<`：小于
- `==`：等于，等值比较
- `=~`：左侧是字符串，右侧是一个模式，判断左侧的字符串能否被右侧的模式所匹配：但是必须在`[[]]`中执行模式匹配。模式中可以使用行首、行尾锚定符，但是模式不要加引号，有时候可能不需要转义
- `!=`，`<>`：不等于
- `-n`: 判断字符串是否不空，不空为真，空则为假
- `-z`：判断字符串是否为空，空则为真，不空则假

执行字符条件测试表达式的操作符有

- `[ 字符条件测试表达式 ]`：前后有空格
- `[[ 字符条件测试表达式 ]]`：前后有空格

0x0202 文件条件测试

组成文件条件测试表达式的操作符有

- `-e file`：测试文件是否存在
- `-a file`：测试文件是否存在
- `-f file`：测试是否为普通文件
- `-d directory`： 测试是否为目录文件
- `-b file`：测试文件是否存在并且是否为一个块设备文件
- `-c file`：测试文件是否存在并且是否为一个字符设备文件
- `-h|-L file`：测试文件是否存在并且是否为符号链接文件
- `-p file`：测试文件是否存在并且是否为管道文件：
- `-S file`：测试文件是否存在并且是否为套接字文件：
- `-r file`：测试其有效用户是否对此文件有读取权限
- `-w file`：测试其有效用户是否对此文件有写权限
- `-x file`：测试其有效用户是否对此文件有执行权限
- `-s file`：测试文件是否存在并且不空
- `file1 -nt file2`：测试file1是否比file2更new一些
- `file1 -ot file2`：测试file1是否比file2更old一些

执行文件条件测试表达式的操作符有

- `[ 文件条件测试表达式 ]`：前后有空格
- `[[ 文件条件测试表达式 ]]`：前后有空格

0x03 逻辑操作符

逻辑操作符有

- 逻辑与`&&`
  - 真 `&&` 真 = 真
  - 真 `&&` 假 = 假
  - 假 `&&` 真 = 假
  - 假 `&&` 假 = 假
- 逻辑或`||`
  - 真 `||` 真 = 真
  - 真 `||` 假 = 真
  - 假 `||` 真 = 真
  - 假 `||` 假 = 假
- 逻辑非`!`
  - ！真 = 假
  - ！假 = 真

注意：各种编译语言对逻辑真、假的定义不同，在shell中，状态值为0代表真，状态值为非0代表假

0x04 括号操作符

括号操作符有以下几种

- `()`
  - 命令组：括号中的命令将会新开一个子shell顺序执行，所以括号中的变量不能够被脚本余下的部分使用；括号中多个命令之间用分号隔开，最后一个命令可以没有分号，各命令和括号之间不必有空格，即`(cmd1;cmd2;cmd3)`
  - 命令替换：等同于`` `cmd` ``，shell扫描一遍命令行，发现了`$(cmd)`结构，便将`$(cmd)`中的cmd执行一次，得到其标准输出，再将此输出放到原来命令。有些shell不支持，例如`tcsh`
  - 数组初始化：用来初始化数组
- `(())`
  - 执行算术表达式：这种算术表达式是整数型的计算，不支持浮点型
  - 执行进制运算：`$((16#5f))`结果为95(16进位转十进制)
  - 重定义变量值：`a=5;((a++))`结果a被重定义为6
  - 算术运算比较：双括号中的变量可以不使用$符号前缀，括号内支持多个表达式用逗号分开；比如可以直接使用`for((i=0;i<5;i++))`
- `[]`
  - 执行测试表达式：前后有空格
  - 执行算术表达式：前后没有空格
- `[[]]`
  - 执行测试表达式：前后有空格
- `{}`

  - 命令组：括号中的命令将会在当前shell顺序执行，所以括号中的变量能被脚本余下的部分使用；括号中多个命令之间用分号隔开，最后一个命令后必须有分号，并且第一条命令和左括号之间必须用空格隔开，即`{ cmd1;cmd2;cmd3;}`
  - 变量引用：`${}`
  - 生成列表：`{a..d}.txt`表示`a.txt`、`b.txt`、`c.txt`、`d.txt`；在括号中，不允许有空白，除非这个空白被引用或转义
  - 扩展：`{a,b}.txt`表示`a.txt`和`b.txt`；在括号中，不允许有空白，除非这个空白被引用或转义

控制流程语句

和其它编程语言一样，shell的控制流程语句大体上也分为三种

- 顺序执行语句
- 条件执行语句
  - *if*条件语句
  - *case*条件语句
  - *select*条件语句
  - 条件测试表达式
    - 整数测试表达式
    - 字符测试表达式
    - 文件测试表达式
    - 其它测试表达式
- 循环执行语句
  - *for*循环语句
  - *while*循环语句
  - *until*循环语句
  - 循环退出命令

0x00 顺序执行语句

顺序执行语句是默认法则，即按照自上而下、自左往右的顺序逐条执行各命令，每执行一次就会得到对应的结果，然后退出该次执行操作

0x01 条件执行语句

条件执行语句会根据判断条件选择符合条件的分支执行对应的`cmd_list`命令列表，执行完命令后就会退出该分支；条件执行语句有以下几种

- *if*条件语句
- *case*条件语句
- *select*条件语句

0x0100 if条件语句

if条件语句的语法结构如下(使用help if命令可以查看)

```
if TEST_COMMANDS; then
        COMMANDS_LIST;
[elif TEST_COMMANDS; then
        COMMANDS_LIST;]
...
[else
        COMMANDS_LIST;]
fi
```

其执行逻辑是

- **1.** 先执行if分支下的TEST_COMMANDS条件测试命令，如果执行完的状态返回值为非0，则执行第2步；如果执行完的状态返回值为0，即TEST_COMMANDS条件测试命令执行成功，则执行该分支下的COMMANDS_LIST命令列表，执行完后就直接退出，此时整个if语句结构体的状态返回值取决于COMMANDS_LIST命令列表中最后一个命令的状态返回值
- **2.**如果存在elif分支，则按照第一步的流程依次执行elif分支下的TEST_COMMANDS条件测试命令，如果没有一个elif分支的状态返回值为0，则执行第3步；如果存在一个elif分支的状态返回值为0，即该分支下的TEST_COMMANDS条件测试命令执行成功，则执行该分支下的COMMANDS_LIST命令列表，执行完后就直接退出，此时整个if语句结构体的状态返回值取决于COMMANDS_LIST命令列表中最后一个命令的状态返回值
- **3.**如果else分支不存在，那么整个if语句结构体的状态返回值为0；如果存在else分支，则执行该分支下的COMMANDS_LIST命令列表，执行完后就直接退出，此时整个if语句结构体的状态返回值取决于COMMANDS_LIST命令列表中最后一个命令的状态返回值

在整个if语句结构体中有两个地方需要注意

- COMMANDS_LIST：表示待执行的命令列表，即一系列shell命令的集合，类型格式多种多样，在一系列示例代码中可见一斑
  - 注意：在命令列表中不能使用()操作符改变优先级，它的作用是让括号内的语句成为命令列表进入子shell中执行，它的具体作用可参考：括号操作符
- TEST_COMMANDS：表示条件测试命令，即通过引用条件测试命令的执行状态返回值是否为0来判断是否执行上述COMMANDS_LIST命令列表；**这里需要特别注意的是，和其它语言不通，shell的条件测试命令只有以下三种类型**
  - 命令执行：命令本身执行后就会产生对应的执行状态返回值，所以可以直接用来做条件判断
    - 此时不能使用“‘操作符来引用命令，因为该操作引用的是命令的执行结果，而不是命令的执行状态返回值
    - 通常是直接使用命令，然后在命令后面添加s&> /dev/null，表示将命令的执行结果重定向至/dev/null，只引用其状态返回值；例如：if grep "^root" /etc/passwd &> /dev/null; then
  - 执行条件测试表达式：在shell中，条件测试表达式是由条件测试操作符以及对应的操作数组成，详细介绍可参考下列：条件测试表达式，执行条件测试表达式有以下三种格式
    - test Test_Expression：通过test命令执行
    - [Test_Expression]：通过[]操作符执行，注意Test_Expression 前后有空格
    - [[Test_Expression]]：通过[[]]操作符执行，注意Test_Expression 前后有空格
  - 组合条件测试：即对多个命令执行状态返回值或者执行条件测试表达式状态返回值做逻辑运算，组合条件测试有以下三种格式
    - 逻辑与操作：只有当&&操作符两边执行结果都为真(状态值为0)，最后组合条件测试结果才为真(状态值为0)

        - `[ Test_Expression1 ] && [ Test_Expression2 ]`：此处使用`[]`或`[[]]`都行
        - `COMMAND &> /dev/null && [ Test_Expression2 ]`：此处使用`[]`或`[[]]`都行
        - `COMMAND1 &> /dev/null && COMMAND2 &> /dev/null &&`
        - `[ Test_Expression1 -a Test_Expression2 ]`：此处使用`[]`或`[[]]`都行
        - `[[ Test_Expression1 && Test_Expression2 ]]`：此处只能使用`[[]]`操作符，因为`&&`运算符不允许用于`[]`操作符中
    * 逻辑或操作：只要`||`操作符两边执行结果有一个为真(状态值为0)，最后组合条件测试结果就为真(状态值为0)
        - `[ Test_Expression1 ] || [ Test_Expression2 ]`：此处使用`[]`或`[[]]`都行
        - `COMMAND &> /dev/null || [ Test_Expression2 ]`：此处使用`[]`或`[[]]`都行
        - `COMMAND1 &> /dev/null || COMMAND2 &> /dev/null &&`
        - `[ Test_Expression1 -o Test_Expression2 ]`：此处使用`[]`或`[[]]`都行
        - `[[ Test_Expression1 || Test_Expression2 ]]`：此处只能使用`[[]]`操作符，因为`||`运算符不允许用于`[]`操作符中
    * 逻辑非操作：对`!`右侧执行结果取反
        - `! [ Test_Expression ]`：此处使用`[]`或`[[]]`都行
        - `! COMMAND1 &> /dev/null`
        - `! ([ Test_Expression1 ] || [ Test_Expression2 ])`：此处相当于`! [ Test_Expression1 ] && ! [ Test_Expression2 ]`
        - `! ([ Test_Expression1 ] && [ Test_Expression2 ])`：此处相当于`! [ Test_Expression1 ] || ! [ Test_Expression2 ]`
    * 注意：**非的优先级大于与，与的优先级大于或**

示例代码如下

- 输出两个传入参数中的最大值

```
#!/bin/bash
if [ $# -lt 2 ]; then
        echo "`basename $0` arg1 arg2"
        exit 1
fi
if [ $1 -gt $2 ]; then
        echo "the max num is $1"
else
        echo "the max num is $2"
fi
```

- 计算1~200之间偶数之和

```
#!/bin/bash

declare -i sum=0
for i in {1..200};do
        if [ $[$i%2] -eq 0 ]; then
                let sum+=$i
        fi
done
```

(continues on next page)

(continued from previous page)

```
echo "the sum is : $sum"
```

- 让用户输入一个用户名，先判断该用户是否存在，不存在，则以7为退出码；如果存在，判断用户的shell是否为/bin/bash，如果是，则显示为Bash User，退出码为0，否则显示为Not Bash User，退出码为1

```
#!/bin/bash

read -p "please input username: " username

echo $username
if ! grep "^$username\>" /etc/passwd &> /dev/null; then
        echo "User not exist"
        exit 7
elif [[ `grep "^$username\>" /etc/passwd | cut -d: -f7` =~ /bin/bash ]];then
        echo "Bash User"
        exit 0
else
        echo "Not Bash User"
        exit 1
fi
```

- 统计输入文件的空白行数

```
#!/bin/bash

read -p "Enter a file path: " filename

if grep "^$" $filename &> /dev/dull; then
        linesCount=`grep "^$" $filename | wc -l`
        echo "$filename has $linesCount space lines"
else
        echo "$filename has no space linse"
fi
```

0x0101 case条件语句

case条件语句的语法结构如下(使用help case命令可以查看)

```
case WORD in
        PATTERN1)
                COMMANDS_LIST
                ;;
        PATTERN2)
                COMMANDS_LIST
                ;;
        PATTERN3)
                COMMANDS_LIST
                ;;
        ...
esac
```

其执行逻辑是：WORD依次匹配PATTERN1、PATTERN2、PATTERN3.....；如果所有模式都没有匹配上，则直接退出case语句，此时执行状态返回值为0；如果匹配上任意一个PATTERN就执行该分支下面的COMMANDS_LIST命令列表，执行完后就直接退出，此时整个case语句结构体的状态返回值取决于COMMANDS_LIST命令列表中最后一个命令的状态返回值；模式的匹配优先级是PATTERN1> PATTERN2> PATTERN3> ......

在以上结构中，有以下几点需要注意

- case中的每个小分支都以双分号`;;`结尾，表示执行完该分支后直接退出case语句；但最后一个小分句的双分号可以省略。实际上，小分句除了使用`;;`结尾，还可以使用`;&`和`;;&`结尾，只不过意义不同，如下
  - `;;`符号表示小分支执行完成后立即退出case语句
  - `;&`符号表示继续执行下一个小分支中的COMMANDS_LIST部分，而无需进行匹配动作，并由此小分支的结尾符号来决定是否继续操作下一个小分句
  - `;;&`符号表示继续向后(不止是下一个，而是一直向后)匹配小分支，如果匹配成功，则执行对应小分支中的COMMANDS_LIST部分，并由此小分支的结尾符号来决定是否继续向后匹配
- 每个小分支中的PATTERN部分都使用括号`()`包围，只不过左括号`(`不是必须的
- 一般最后一个小分支使用的PATTERN是`*`，表示无法匹配前面所有小分支时，将匹配该小分支；用来避免case语句无法匹配的情况，在shell脚本中，此小分支一般用于提示用户脚本的使用方法，即给出脚本的Usage

这里也需要说明下以下两个关键组成成分

- WORD：一般是字符串类型
- PATTERN：该模式支持通配符机制(注意不是正则表达式)
  - `*`：匹配任意长度的任意字符
  - `?`: 匹配单个任意字符
  - `[]`: 匹配指定字符范围内的任意单个字符，不区分大小写
    - `[a-z]`：不区分大小写，可以匹配大写字母
    - `[A-Z]`：不区分大小写，可以匹配小写字母
    - `[0-9]`：匹配0到9任意单个数字
    - `[a-z0-9]`：匹配单个字母或数字
    - `[[:upper:]]`：匹配单个大写字母
    - `[[:lower:]]`：匹配单个小写字母
    - `[[:alpha:]]`：匹配单个大写或小写字母
    - `[[:digit:]]`：匹配单个数字
    - `[[:alnum:]]`：匹配单个字母或数字
    - `[[:space:]]`：匹配单个空格字符
    - `[[:punct:]]`：匹配单个标点符号
  - `[^]`: 匹配指定字符范围外的任意单个字符
    - `[^a-z]`：匹配字母之外的单个字符
    - `[^A-Z]`：匹配字母之外的单个字符
    - `[^0-9]`：匹配数字之外的单个字符
    - `[^a-z0-9]`：匹配字母和数字之外的单个字符
    - `[^[:upper:]]`：匹配大写字母之外的单个字符
    - `[^[:lower:]]`：匹配小写字母之外的单个字符
    - `[^[:alpha:]]`：匹配字母之外的单个字符
    - `[^[:digit:]]`：匹配数字之外的单个字符
    - `[^[:alnum:]]`：匹配字母和数字之外的单个字符
    - `[^[:space:]]`：匹配空格字符之外的单个字符
    - `[^[:punct:]]`：匹配标点符号之外的单个字符

  - |：用来分隔上述* 、?、[]、[^]通配元字符；例如([yY] | [yY][eE][sS])表示即可以输入单个字母的y或Y，还可以输入yes三个字母的任意大小写格式

示例代码如下

```
#!/bin/bash
set -- y

case "$1" in
    ([yY]|[yY][eE][sS])
        echo yes;&
    ([nN]|[nN][oO])
        echo no;;
    (*)
        echo wrong;;
esac

# 执行结果如下
# yes
# no
```

其中set -- string_list的作用是将string_list按照IFS分隔后分别赋值给位置变量$1、$2、$3...，因此此处是为$1赋值字符y

在此示例中，$1能匹配第一个小分支，但第一个小分支的结尾符号为;&，所以无需判断地直接执行第二个小分支的echo no，但第二个小分支的结尾符号为;;，于是直接退出case语句。因此，即使$1无法匹配第二个小分句，case语句的结果中也输出了yes和no

```
#!/bin/bash
set -- y

case "$1" in
    ([yY]|[yY][eE][sS])
        echo yes;;&
    ([nN]|[nN][oO])
        echo no;;
    (*)
        echo wrong;;
esac

# 执行结果如下
# yes
# wrong
```

在此示例中，$1能匹配第一个小分支，但第一个小分支的结尾符号为;;&，所以继续向下匹配，第二个小分支未匹配成功，直到第三个小分支才被匹配上，于是执行第三个小分支中的echo wrong，但第三个小分支的结尾符号为;;，于是直接退出case语句。所以，结果中输出了yes和wrong

0x0102 select条件语句

select条件语句是一种可以提供菜单选择的条件判断语句，其语法结构如下(使用help select命令可以查看)

```
select NAME [in WORDS ... ;] do
        COMMANDS_LIST
done
```

其执行逻辑是

- **1.**如果in WORDS部分存在，则会将WORDS部分根据环境变量IFS进行分割，对分割后的每一项依次进行编号作为菜单项输出；如果in WORDS部分不存在，则使用in $@代替，即将位置变量的内容进行编号作为菜单项输出

- **2.**当输入内容能够匹配输出菜单序号时，该序号将会保存到变量NAME中，该序号对应的内容将会保存到特殊变量REPLY中；当输入内容不能匹配输出菜单序号时，比如随便几个字符，变量NAME将会被置空，特殊变量REPLY将会保存所有输入内容
- **3.**每次输入选择保存NAME和REPLY变量后，就会直接执行COMMANDS_LIST部分；如果没有break命令，则会跳回第一步，循环重复执行，直到遇到break命令或者ctrl+c退出select语句

示例代码如下

```
#!/bin/bash

select fname in cat dog sheep mouse;do
        echo your choice: \"$REPLY\) $fname\"
done

# 执行结果如下
[root@localhost ~]# ./test.sh
1) cat
2) dog
3) sheep
4) mouse
#? 1                        # 输入序号1
your choice: "1) cat"
#? 2                        # 输入序号2
your choice: "2) dog"
#? 3                        # 输入序号3
your choice: "3) sheep"
#? 4                        # 输入序号4
your choice: "4) mouse"
#? 5                        # 输入序号5，没有该序号值，所有fname变量置空
your choice: "5) "
#? anony                    # 输入anony，不是序号值，所以fname变量置空
your choice: "anony) "
#? ^C                       # select语句中没有break命令，通过ctrl+c退出select语句
```

0x0103 条件测试表达式

条件测试表达式有以下几种类型

- 整数测试表达式
- 字符测试表达式
- 文件测试表达式
- 其它测试表达式

整数测试表达式的格式为：`NUM1 操作符 NUM2`

- NUM1和NUM2是整数，可以直接是整数值(例如：2)，可以是变量引用(例如：$#)，也可以是算术运算得到的值(参考算术运算)
- 整数测试操作符有
  - -eq：等于
  - -ne：不等于
  - -le：小于等于
  - -ge：大于等于
  - -lt：小于
  - -gt:大于

字符测试表达式的格式有两种格式

- 双目测试格式：`STR1 双目操作符 STR2`
  - `STR1`和`STR2`是字符串，shell中默认数据类型是字符串，即不带""默认都会被当做字符串类型；但是在此处，必须使用""(除非是模式匹配中的模式字符串，才不用引号)
  - 双目测试操作符有
    - `>`：表示左边的字符串大于右边的字符串
    - `<`：表示左边的字符串小于右边的字符串
    - `==`：表示左边的字符串等于右边的字符串
    - `!=`、`<>`：表示左右两边的字符串完全不相等
    - `=~`：左侧是普通字符串，右侧是一个模式字符串，用来判断左侧的字符串能否被右侧的模式所匹配：但是必须在`[[]]`中才能执行模式匹配；模式中可以使用行首、行尾锚定符，但是**模式不要加引号**，有时候可能不需要转义，具体模式书写格式可参考正则表达式
- 单目测试格式：单目操作符 `STR`
  - `STR`是字符串，shell中默认数据类型是字符串，即不带""默认都会被当做字符串类型；但是在此处，必须使用""
  - 单目测试操作符有
    - `-n`: 判断字符串是否不空，不空为真，空则为假
    - `-z`：判断字符串是否为空，空则为真，不空则假

文件测试表达式的格式也有两种

- 单目测试格式：单目操作符 `FILE`
  - `FILE`是文件名，一般使用绝对路径
  - 单目操作符有
    - `-e FILE`：测试文件是否存在
    - `-a FILE`：测试文件是否存在
    - `-f FILE`：测试是否为普通文件
    - `-d FILE`： 测试是否为目录文件
    - `-b FILE`：测试文件是否存在并且是否为一个块设备文件
    - `-c FILE`：测试文件是否存在并且是否为一个字符设备文件
    - `-h|-L FILE`：测试文件是否存在并且是否为符号链接文件
    - `-p FILE`：测试文件是否存在并且是否为管道文件：
    - `-S FILE`：测试文件是否存在并且是否为套接字文件：
    - `-r FILE`：测试其有效用户是否对此文件有读取权限
    - `-w FILE`：测试其有效用户是否对此文件有写权限
    - `-x FILE`：测试其有效用户是否对此文件有执行权限
    - `-s FILE`：测试文件是否存在并且不空
- 双目测试格式：`FILE1 双目操作符 FILE2`
  - `FILE1`和`FILE2`是文件名，一般使用绝对路径
  - 双目操作符有
    - `FILE1 -nt FILE2`：测试FILE1是否比FILE2更new一些
    - `FILE1 -ot FILE2`：测试FILE1是否比FILE2更old一些

其它测试表达式的格式有

- 整型值
  - [0]表示真
  - [非0]表示假

0x02 循环执行语句

循环执行语句会根据判断条件循环多次执行对应的循环体cmd_list命令列表，当判断条件不满足时就会退出该循环体，需要注意的是：**循环必须有退出条件，否则将陷入死循环**；循环执行语句有以下几种

- *for*循环语句
- *while*循环语句
- *until*循环语句

0x0200 for循环语句

for循环语句在shell脚本中应用及其广泛，它有两种语法结构(使用help for命令可以查看)

```
# 结构一
for NAME [in WORDS ... ] ; do
        COMMANDS_LIST
done


# 结构二
for (( exp1; exp2; exp3 )); do
        COMMANDS_LIST
done
```

语法结构一的执行逻辑是

- **1.**如果in WORDS部分存在，则会将WORDS部分根据环境变量IFS进行分割，依次赋值给变量NAME(**如果WORD中使用引用包围了某些单词，则将引号包围的内容分隔为一个单词**)；如果in WORDS部分不存在，则默认使用in $@代替，即将位置变量依次赋值给变量NAME
- **2.**NAME变量每被赋值一次，就会执行一次循环体COMMANDS_LIST，直到第一步中所有分隔部分给NAME变量赋值完毕，才会结束循环
- **3.**如果在循环体中遇到continue命令，则退出当前for循环，直接进行下一for循环；如果遇到break命令，则直接退出for循环结构体
- **4.**整个for语句结构体的状态返回值取决于退出整个for循环结构体时最后一个命令的执行状态返回值

语法结构二的执行逻辑是

- **1.**首先执行算术表达式exp1
- **2.**然后判定算术表达式exp2的状态返回值是否为0，如果为0则执行循环体COMMANDS_LIST，执行完之后，执行算术表达式exp3，然后再次判定算术表达式exp2的状态返回值是否为0；直到其状态返回值为非0才退出整个for循环结构体，否则就会循环执行第2步，此时整个for循环的状态返回值为退出整个for循环结构体时最后一个算术表达式exp2的状态返回值
- **3.**如果在循环体中遇到continue命令，则退出当前for循环，直接进行下一for循环(即直接执行上述第二步)，此时整个for循环的状态返回值为退出整个for循环结构体时最后一个算术表达式exp2的状态返回值；如果遇到break命令，则直接退出整个for循环结构体，此时整个for语句结构体的状态返回值取决于退出整个for循环结构体时最后一个命令的执行状态返回值

for循环语句的循环退出机制有：

- continue：跳出当前循环进入下一循环
- break[n]：默认跳出整个循环；n可以指定跳出几层循环
- 列表遍历：使用一个变量去遍历给定列表中的每个元素(以环境变量IFS为分隔符)，在每次变量赋值时执行一次循环体，直至赋值完成所有元素退出循环
- 算术执行：引用算术表达式的执行状态返回值来判断是否退出整个循环

for循环语句适用于已知循环次数的场景

语法结构一中的WORDS有多种表现形式

- 列表变量
  - 数字列表：数字列表示例代码
    - {start..end}
    - `seq start step end`
  - 其它列表：其它列表示例代码
    - 使用空白分隔符直接给出列表
    - 使用文件名通配机制生成列表
    - 使用命令生成列表
- 数组变量
  - 普通数组：普通数组示例代码
  - 关联数组：关联数组示例代码

语法结构二种的exp只支持数学计算和比较，因为它被包含在执行算术运算的(())操作符之内

- exp1：一般是赋值表达式，例如for ((i=1,j=3;i<=3 && j>=2;++i,--j));do echo $i $j;done
- exp2：一般是比较表达式，例如for ((i=1,j=3;i<=3 && j>=2;++i,--j));do echo $i $j;done，比较表达式可参考数值类型比较运算for循环部分
- exp3：一般是计算表达式，例如for ((i=1,j=3;i<=3 && j>=2;++i,--j));do echo $i $j;done，计算表达式可参考数值类型算术运算

示例代码如下

- 计算当前系统所有用户ID之和

```
#!/bin/bash

declare -i uidSum=0

for i in `cut -d: -f3 /etc/passwd`; do
        uidSum=$[$uidSum+$i]
done

echo "the UIDSum is: $uidSum"
```

- 新建用户tmpuser1-tmpuser10，并计算它们的id之和

```
#!/bin/bash

declare -i uidSum=0

for i in {1..10}; do
        useradd tmpuser$i
```

(continues on next page)

(continued from previous page)

```
        let uidSum+=`id -u tmpuser$i`
done

echo "the UIDSum is: $uidSum"
```

- 输出1-10之间的所有偶数

```
#!/bin/bash

for ((i=1;i<=10;i++));do
        let tmp=i%2
        if [ $tmp -eq 0 ]; then
                echo $i
        fi
done
```

0x0201 while循环语句

while循环语句的语法结构如下(使用help while命令可以查看)

```
while TEST_COMMANDS_LIST; do
        COMMANDS_LIST
done
```

其执行逻辑是

- **1.**先执行TEST_COMMANDS_LIST条件测试命令，如果其最后一个命令的执行状态返回值为0，则执行循环体COMMANDS_LIST，执行完后，再次执行TEST_COMMANDS_LIST条件测试命令，直到其最后一个名的状态返回值为非0才会退出整个while循环体，否则将一直循环执行该步，此时整个while循环的状态返回值为退出循环结构体时最后一个TEST_COMMANDS_LIST条件测试命令的最后一个命令的状态返回值
- **2.**如果在循环体中遇到continue命令，则退出当前while循环，直接进行下一while循环(即直接执行上述第一步)，此时整个while循环的状态返回值为退出循环结构体时最后一个TEST_COMMANDS_LIST条件测试命令的最后一个命令的状态返回值；如果遇到break命令，则直接退出整个while循环结构体，此时整个while语句结构体的状态返回值取决于退出整个循环结构体时最后一个命令的执行状态返回值

在上述while循环语句结构中需要注意的是

- COMMANDS_LIST：表示待执行的命令列表(也称为while循环体)，即一系列shell命令的集合，类型格式多种多样，在一系列示例代码中可见一斑
  - 注意：在命令列表中不能使用()操作符改变优先级，它的作用是让括号内的语句成为命令列表进入子shell中执行，它的具体作用可参考：括号操作符
- TEST_COMMANDS_LIST：表示条件测试命令，即通过引用条件测试命令的执行状态返回值是否为0来判断是否执行上述COMMANDS_LIST循环体；**这里需要特别注意的是，和其它语言不通，shell的条件测试命令只有以下三种类型**
  - 命令执行：命令本身执行后就会产生对应的执行状态返回值，所以可以直接用来做条件判断
    - 此时不能使用‘‘操作符来引用命令，因为该操作引用的是命令的执行结果，而不是命令的执行状态返回值
    - 通常是直接使用命令，然后在命令后面添加s&> /dev/null，表示将命令的执行结果重定向至/dev/null，只引用其状态返回值；例如：if grep "^root" /etc/passwd &> /dev/null; then
  - 执行条件测试表达式：在shell中，条件测试表达式是由条件测试操作符以及对应的操作数组成，详细介绍可参考下列：条件测试表达式，执行条件测试表达式有以下三种格式

    * test Test_Expression：通过test命令执行
    * [Test_Expression]：通过[]操作符执行，注意Test_Expression 前后有空格
    * [[Test_Expression]]：通过[[]]操作符执行，注意Test_Expression 前后有空格
  - 组合条件测试：即对多个命令执行状态返回值或者执行条件测试表达式状态返回值做逻辑运算，组合条件测试有以下三种格式
    * 逻辑与操作：只有当&&操作符两边执行结果都为真(状态值为0)，最后组合条件测试结果才为真(状态值为0)
      · [Test_Expression1] && [Test_Expression2]：此处使用[]或[[]]都行
      · COMMAND &> /dev/null && [Test_Expression2]：此处使用[]或[[]]都行
      · COMMAND1 &> /dev/null && COMMAND2 &> /dev/null &&
      · [Test_Expression1 -a Test_Expression2]：此处使用[]或[[]]都行
      · [[Test_Expression1 && Test_Expression2]]：此处只能使用[[]]操作符，因为&&运算符不允许用于[]操作符中
    * 逻辑或操作：只要||操作符两边执行结果有一个为真(状态值为0)，最后组合条件测试结果就为真(状态值为0)
      · [Test_Expression1] || [Test_Expression2]：此处使用[]或[[]]都行
      · COMMAND &> /dev/null || [Test_Expression2]：此处使用[]或[[]]都行
      · COMMAND1 &> /dev/null || COMMAND2 &> /dev/null &&
      · [Test_Expression1 -0 Test_Expression2]：此处使用[]或[[]]都行
      · [[Test_Expression1 || Test_Expression2]]：此处只能使用[[]]操作符，因为||运算符不允许用于[]操作符中
    * 逻辑非操作：对!右侧执行结果取反
      · ! [Test_Expression]：此处使用[]或[[]]都行
      · ! COMMAND1 &> /dev/null
      · ! ([Test_Expression1] || [Test_Expression2])：此处相当于! [Test_Expression1] && ! [Test_Expression2]
      · ! ([Test_Expression1] && [Test_Expression2])：此处相当于! [Test_Expression1] || ! [Test_Expression2]
    * 注意：**非的优先级大于与，与的优先级大于或**

while循环语句的循环退出机制有：

- continue：跳出当前循环进入下一循环
- break[n]：默认跳出整个循环；n可以指定跳出几层循环
- 条件测试：此时为了避免死循环，TEST_COMMANDS_LIST条件测试里必须有控制循环次数的变量；COMMANDS_LIST循环体里必须有改变条件测试中用于控制循环次数变量的值操作

while循环语句适用于循环次数未知的场景，示例代码如下

```
#!/bin/bash

let i=1,sum=0

```

(continues on next page)

(continued from previous page)

```
# 此处TEST_COMMANDS_LIST有多个命令
# 需要注意的是[ $i -le 10 ]才是判定是否退出循环的命令
# 而echo $i命令的执行状态返回结果跟退出循环无关
while echo $i;[ $i -le 10 ]; do
        let sum=sum+i;
        let ++i
done

echo $sum
```

对于while循环，有另外两种常见的用法

- 实现无限死循环

```
# 格式一：TEST_COMMANDS_LIST部分使用:
while :; do
        COMMANDS_LIST
done

# 格式二：TEST_COMMANDS_LIST部分使用true
while true; do
        COMMANDS_LIST
done
```

- 实现read命令从标准输入中按行读取值，然后保存到变量line中(既然是read命令，就可以保存到多个变量中)，读取一行就是一个循环

```
##############################方法一#######################################
# 标准输入来自于管道
# 每读取一行内容就会进入一次while循环，此处有两行内容所以进行两次while循环
# 此处通过-e选项实现多行输入
# 读取的每行内容将会按照IFS分隔，并赋值给两个变量
declare -i linenum=0
echo -e "abc xyz\n2abc 2xyz" | while read field1 field2; do
        echo $field1
        echo $field2
        linenum+=1
done
echo "there are $linenum lines"
# 此处使用的是管道符号，这样使得while语句在子shell中执行，这也意味着while语句内部设置的变量、数组、函数等在while循环外部都不再生效
# 执行结果如下
# abc
# xyz
# 2abc
# 2xyz
# there are 0 lines


##############################方法二#######################################
# 标准输入来自于重定向
# 每读取一行内容就会进入一次while循环，此处有两行内容所以进行两次while循环
# 此处通过EOF标志实现多行输入
# 读取的每行内容将会按照IFS分隔，并赋值给两个变量
declare -i linenum=0
while read field1 field2; do
        echo $field1
        echo $field2
        linenum+=1
done << EOF
abc xyz
```

(continues on next page)

(continued from previous page)

```
2abc 2xyz
EOF
echo "there are $linenum lines"
# 此处while语句内部设置的变量、数组、函数等在while循环外部依然生效
# 执行结果如下
# abc
# xyz
# 2abc
# 2xyz
# there are 2 lines


##############################方法三#####################################
# 标准输入来自于重定向
# 常用来重定向文件输入，读取文件内容
# 每读取文件一行内容，就会进入一次while循环，直到读完文件尾部退出循环
while read line; do
        echo $line
done < /etc/passwd


##############################方法四#####################################
# 读取文件的另一种写法
exec </etc/passwd;while read line; do
echo $line
done
```

关于read命令从标准输入中按行读取值的几种while循环的写法，还有一点需要注意

- 方法一传递数据的源是一个单独的进程，它传递的数据只要被while循环读取一次，所有剩余的数据就会被丢弃
- 方法二、三、四是以实体文件作为重定向传递的数据，while循环读取一次之后并不会丢弃剩余数据，直到数据完全读取完毕

也就是说当标准输入是非实体文件时(如管道传递、独立进程产生的)只供一次读取；当标准输入是直接重定向实体文件时，可供多次读取，但只要某一次读取了该文件的全部内容就无法再提供读取

回到IO重定向上，无论什么数据资源，只要被读取完毕或者主动丢弃，那么该资源就不可再得

- 对于独立进程传递的数据(管道左侧进程产生的数据、进程替换产生的数据)，它们都是虚拟数据，要不被一次读取完毕，要不读一部分剩余的丢弃，这是真正的一次性资源；其实这也是进程间通信时数据传递的现象
- 实体文件重定向传递的数据，只要不是一次性被全部读取，它就是可再得资源，直到该文件数据全部读取结束，这是伪一次性资源

大多数情况下，独立进程传递的数据和文件直接传递的数据并没有什么区别，但有些命令可以标记当前读取到哪个位置，使得下次该命令的读取动作可以从标记位置处恢复并继续读取，特别是这些命令用在循环中时。这样的命令有`head -n N`和`grep -m`，经测试，`tail`并没有位置标记的功能，因为`tail`读取的是后几行，所以它必然要读取到最后一行并计算要输出的行，所以`tail`的性能比`head`要差

- 示例一：通过管道将实体文件的内容传递给head

```
#!/bin/bash
declare -i i=0

cat /etc/passwd | while head -n 2; [[ $i -le 3 ]]; do
        echo $i
        let ++i
done
```

(continues on next page)

(continued from previous page)

```
# 执行结果如下
# root:x:0:0:root:/root:/bin/bash
# bin:x:1:1:bin:/bin:/sbin/nologin
# 0
# 1
# 2
# 3
```

- 示例二：将实体文件重定向传递给head

```
#!/bin/bash
declare -i i=0

while head -n 2; [[ $i -le 3 ]]; do
        echo $i
        let ++i
done < /etc/passwd

# 执行结果如下
# root:x:0:0:root:/root:/bin/bash
# bin:x:1:1:bin:/bin:/sbin/nologin
# 0
# daemon:x:2:2:daemon:/sbin:/sbin/nologin
# adm:x:3:4:adm:/var/adm:/sbin/nologin
# 1
# lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
# sync:x:5:0:sync:/sbin:/bin/sync
# 2
# shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
# halt:x:7:0:halt:/sbin:/sbin/halt
# 3
# mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
# operator:x:11:0:operator:/root:/sbin/nologin
```

分析上述结果可以看到

- 示例一中：本该head应该每次读取2行，但实际执行结果中显示总共就只读取了2行
- 示例二中：head每次读取2行，而且每次读取的两行是不同的，后一次读取的两行是从前一次读取结束的地方开始的，这是因为head有读取到指定行数后做上位置标记的功能

要想确定命令、工具是否具有做位置标记的能力，只需像下面例子一样做个简单的测试。以head和sed为例，即使sed的q命令能让sed匹配到内容就退出，但却不做位置标记，而且数据资源使用一次就丢弃

```
[root@localhost ~]# (head -n 2;head -n 2) </etc/fstab

#
# /etc/fstab
# Created by anaconda on Sun May 27 09:35:43 2018
[root@localhost ~]# (sed -n /default/'{p;q}' ;sed -n /default/'{p;q}') </etc/fstab
/dev/mapper/centos-root /                       xfs     defaults        0 0
[root@localhost ~]#
```

其实在实际应用过程中，这根本就不是个问题，因为搜索和处理文本数据的工具虽然不少，但绝大多数都是用一次文本就丢一次，几乎不可能因此而产生问题。之所以说这么多废话，主要是想说上面的read读取数据while写法中，管道传递数据是使用最广泛的写法，但其实也是最烂的一种

0x0202 until循环语句

until循环语句的语法结构如下(使用help until命令可以查看)

```
until TEST_COMMANDS_LIST; do
        COMMANDS_LIST
done
```

until循环和while循环的执行思路大致相同，只不过效果相反

- **1.**先执行TEST_COMMANDS_LIST条件测试命令，如果其最后一个命令的执行状态返回值为非0，则执行循环体COMMANDS_LIST，执行完后，再次执行TEST_COMMANDS_LIST条件测试命令，直到其最后一个命令的状态返回值为0才会退出整个until循环体，否则将一直循环执行该步，此时整个until循环的状态返回值为退出循环结构体时最后一个TEST_COMMANDS_LIST条件测试命令的最后一个命令的状态返回值
- **2.**如果在循环体中遇到continue命令，则退出当前until循环，直接进行下一until循环(即直接执行上述第一步)，此时整个until循环的状态返回值为退出循环结构体时最后一个TEST_COMMANDS_LIST条件测试命令的最后一个命令的状态返回值；如果遇到break命令，则直接退出整个until循环结构体，此时整个until语句结构体的状态返回值取决于退出整个循环结构体时最后一个命令的执行状态返回值

在上述until循环语句结构中需要注意的是

- COMMANDS_LIST：表示待执行的命令列表(也称为until循环体)，即一系列shell命令的集合，类型格式多种多样，在一系列示例代码中可见一斑
  - 注意：在命令列表中不能使用()操作符改变优先级，它的作用是让括号内的语句成为命令列表进入子shell中执行，它的具体作用可参考：括号操作符
- TEST_COMMANDS_LIST：表示条件测试命令，即通过引用条件测试命令的执行状态返回值是否为0来判断是否执行上述COMMANDS_LIST循环体；**这里需要特别注意的是，和其它语言不通，shell的条件测试命令只有以下三种类型**
  - 命令执行：命令本身执行后就会产生对应的执行状态返回值，所以可以直接用来做条件判断
    - 此时不能使用“操作符来引用命令，因为该操作引用的是命令的执行结果，而不是命令的执行状态返回值
    - 通常是直接使用命令，然后在命令后面添加s&> /dev/null，表示将命令的执行结果重定向至/dev/null，只引用其状态返回值；例如：if grep "^root" /etc/passwd &> /dev/null; then
  - 执行条件测试表达式：在shell中，条件测试表达式是由条件测试操作符以及对应的操作数组成，详细介绍可参考下列：条件测试表达式，执行条件测试表达式有以下三种格式
    - test Test_Expression：通过test命令执行
    - [Test_Expression]：通过[]操作符执行，注意Test_Expression 前后有空格
    - [[Test_Expression]]：通过[[]]操作符执行，注意Test_Expression 前后有空格
  - 组合条件测试：即对多个命令执行状态返回值或者执行条件测试表达式状态返回值做逻辑运算，组合条件测试有以下三种格式
    - 逻辑与操作：只有当&&操作符两边执行结果都为真(状态值为0)，最后组合条件测试结果才为真(状态值为0)
      - [Test_Expression1] && [Test_Expression2]：此处使用[]或[[]]都行
      - COMMAND &> /dev/null && [Test_Expression2]：此处使用[]或[[]]都行
      - COMMAND1 &> /dev/null && COMMAND2 &> /dev/null &&
      - [Test_Expression1 -a Test_Expression2]：此处使用[]或[[]]都行
      - [[Test_Expression1 && Test_Expression2]]：此处只能使用[[]]操作符，因为&&运算符不允许用于[]操作符中

    * 逻辑或操作：只要||操作符两边执行结果有一个为真(状态值为0)，最后组合条件测试结果就为真(状态值为0)
        - `[ Test_Expression1 ] || [ Test_Expression2 ]`：此处使用[]或[[]]都行
        - `COMMAND &> /dev/null || [ Test_Expression2 ]`：此处使用[]或[[]]都行
        - `COMMAND1 &> /dev/null || COMMAND2 &> /dev/null &&`
        - `[ Test_Expression1 -0 Test_Expression2 ]`：此处使用[]或[[]]都行
        - `[[ Test_Expression1 || Test_Expression2 ]]`：此处只能使用[[]]操作符，因为||运算符不允许用于[]操作符中
    * 逻辑非操作：对!右侧执行结果取反
        - `! [ Test_Expression ]`：此处使用[]或[[]]都行
        - `! COMMAND1 &> /dev/null`
        - `! ([ Test_Expression1 ] || [ Test_Expression2 ])`：此处相当于`! [ Test_Expression1 ] && ! [ Test_Expression2 ]`
        - `! ([ Test_Expression1 ] && [ Test_Expression2 ])`：此处相当于`! [ Test_Expression1 ] || ! [ Test_Expression2 ]`
    * 注意：**非的优先级大于与，与的优先级大于或**

`until`循环语句的循环退出机制有：

- `continue`：跳出当前循环进入下一循环
- `break[n]`：默认跳出整个循环；n可以指定跳出几层循环
- 条件测试：此时为了避免死循环，`TEST_COMMANDS_LIST`条件测试里必须有控制循环次数的变量；`COMMANDS_LIST`循环体里必须有改变条件测试中用于控制循环次数变量的值操作

until循环语句也是适用于循环次数未知的场景，示例代码如下

```
#!/bin/bash

declare -i i=5

until echo hello;[ "$i" -eq 1 ]; do
        let --i
        echo $i
done

# 执行结果如下
# hello
# 4
# hello
# 3
#hello+
# 2
# hello
# 1
# hello
```

0x0203 循环退出命令

循环退出命令有

- `continue [n]`：表示退出当前循环进入下一次循环，适用于`for`、`while`、`until`、`select`语句；n表示退出的循环的次数，默认n=1

- break [n]：表示退出整个循环，适用于for、while、until、select语句；n表示退出的循环层数，默认n=1
- return [n]：表示退出整个函数，适用于函数体内的for、while、until、select语句，同样也适用于函数体内的if、case语句；数值n表示函数的退出状态码，如果没有定义退出状态码，则函数的状态退出码为函数的最后一条命令的执行状态返回值
- exit [n]：表示退出当前shell，适用于脚本的任何地方，表示退出整个脚本；数值n表示脚本的退出状态码，如果没有定义退出状态码，则脚本的状态退出码为脚本的最后一条命令的执行状态返回值

函数

在编程语言中，函数是能够实现模块化编程的工具，每个函数都是一个功能组件，但是函数必须被调用才能执行

函数存在的主要作用在于：最大化代码重用，最小化代码冗余

在shell中，函数可以被当做命令一样执行，它的本质是命令的组合结构体，即可以将函数看成一个普通命令或一个小型脚本。接下来本章内容将从以下几个方面来介绍函数

0x00 函数定义

在shell中函数定义的方法有两种(使用help function命令可以查看)

```
# 方法一
function FuncName {
        COMMANDS_LIST
} [&>/dev/null]

# 方法二
FuncName() {
        COMMANDS_LIST
} [&>/dev/null]
```

上面两种函数定义方法定义了一个名为FuncName的函数

- 方法一中：使用了function关键字，此时函数名FuncName后面的括号可以省略
- 方法二中：省略了function关键字，此时函数名FuncName后面的括号不能省略

COMMANDS_LIST是函数体，它与以下特点

- 函数体通常使用大括号{}包围，由于历史原因，在shell中大括号本身也是关键字，所以为了不产生歧义，函数体和大括号之间必须使用空格、制表符、换行符分隔开来；一般我们都是通过换行符进行分隔
- 函数体中的每一个命令必须使用;或换行符进行分隔；如果使用&结束某条命令，则表示该条命令会放入后台执行

需要注意的是

- &>/dev/null表示将函数体执行过程中可能输出的信息重定向至/dev/null中，该功能可选
- 定义函数时，还可以指定可选的函数重定向功能，这样当函数被调用的时候，指定的重定向也会被执行

- 当前shell定义的函数只能在当前shell使用，子shell无法继承父shell的函数定义，除非使用`export -f`将函数导出为全局函数；如果想取消函数的导出可以使用`export -n`
- 定义了函数后，可以使用`unset -f`移除当前shell中已定义的函数
- 可以使用`typeset -f [func_name]`或`declare -f [func_name]`查看当前shell已定义的函数名和对应的定义语句；使用`typeset -F`或`declare -F`则只显示当前shell中已定义的函数名
- 只有先定义了函数，才可以调用函数；不允许函数调用语句在函数定义语句之前
- 在shell脚本中，函数没有形参的概念，使用方法二定义函数时，括号里什么都不用写，只需要在函数体内使用相关的调用机制调用接收参数即可

0x01 函数调用

函数的调用格式如下

```
FuncName ARGS_LIST
```

其中

- `FuncName`：表示被调用函数的函数名，需要注意的是在shell中函数调用时函数名后面没有`()`操作符
- `ARGS_LIST`：表示被调用函数的传入参数，在shell中给函数传入参数和脚本接收参数的方法相似，直接在函数名后面加上需要传入的参数即可

函数调用时需要注意以下几点

- 如果函数名和命令名相同，则优先执行函数，除非使用`command`命令。例如：定义了一个名为`rm`的函数，在bash中输入`rm`执行时，执行的是`rm`函数，而非`/bin/rm`命令，除非使用`command rm ARGS`，表示执行的是`/bin/rm`命令
- 如果函数名和命令别名相同，则优先执行命令别名，即在优先级方面：别名别名>函数>命令自身

当函数调用函数被执行时，它的执行逻辑如下

- 接收参数：shell函数也接受位置参数变量，但函数的位置参数是调用函数时传递给函数的，而非传递给脚本的参数，所以脚本的位置变量和函数的位置变量是不同的；同时shell函数也接收特殊变量。函数体内引用位置参数和特殊变量方式如下
  - 位置参数
    - `$0`：和脚本位置参数一样，引用脚本名称
    - `$1`：引用函数的第1个传入参数
    - `$n`：引用函数的第n个传入参数
  - 特殊变量
    - `$?`：引用上一条命令的执行状态返回值，状态用数字表示`0-255`
      - `0`：表示成功
      - `1-255`：表示失败；其中`1/2/127/255`是系统预留的，写脚本时要避开与这些值重复
    - `$$`：引用当前shell的PID。除了执行bash命令和shell脚本时，$$不会继承父shell的值，其他类型的子shell都继承
    - `$!`：引用最近一次执行的后台进程PID，即运行于后台的最后一个作业的PID
    - `$#`：引用函数所有位置参数的个数
    - `$*`：引用函数所有位置参数的整体，即所有参数被当做一个字符串
    - `$@`：引用函数所有单个位置参数，即每个参数都是一个独立的字符串
- 执行函数体：在函数体执行时，需要注意的是

  - 函数内部引用变量的查找次序：内层函数自己的变量>外层函数的变量>主程序的变量>bash内置的环境变量
  - 函数内部引用变量的作用域
    * 本地变量：函数体引用本地变量时，重新赋值会覆盖原来的值，如果不想覆盖值，可以使用local进行修饰
    * 局部变量：函数体引用局部变量时，函数退出，将会被撤销
    * 环境变量：函数体引用环境变量时，重新赋值会覆盖原来的值，如果不想覆盖值，可以使用local进行修饰
    * 位置变量：函数体引用位置变量表示引用传递给函数的参数
    * 特殊变量
- 函数返回：函数返回值可分为两类
  - 执行结果返回值：正常的执行结果返回值有以下几种
    * 函数中的打印语句：如echo、print等
    * 最后一条命令语句的执行结果值
    * 和命令一样，函数的执行结果返回值使用“引用
  - 执行状态返回值：执行状态返回值主要有以下几种
    * 使用return语句自定义返回值，即return n，n表示函数的退出状态码，不给定状态码时默认状态码为0
    * 取决于函数体中最后一条命令语句的执行状态返回值
    * 和命令一样，函数的执行状态返回值通过$?引用；或者在条件测试时直接当做命令执行引用(注意此时不能使用[$?]引用函数状态返回值来做条件测试，因为在条件测试中，[]操作符只能执行整数、文件、字符等测试表达式，不能执行命令)

在shell中不仅可以调用本脚本文件中定义的函数，还可以调用其它脚本文件中定义的函数

- 先使用. /path/to/shellscript或source /path/to/shellscript命令导入指定的脚本文件
- 然后使用相应的函数名调用函数即可

0x02 函数退出命令

函数退出命令有

- return [n]：可以在函数体内的任何地方使用，表示退出整个函数；数值n表示函数的退出状态码
- exit [n]：可以在脚本的任何地方使用，表示退出整个脚本；数值n表示脚本的退出状态码

此处需要注意的是：return并非只能用于function内部

- 如果return在function之外，但在.或者source命令的执行过程中，则直接停止该执行操作，并返回给定状态码n(如果未给定，则为0)
- 如果return在function之外，且不在source或.的执行过程中，则这将是一个错误用法

可能有些人不理解为什么不直接使用exit来替代这时候的return。下面给个例子就能清楚地区分它们

先创建一个脚本文件proxy.sh，内容如下，用于根据情况设置代理的环境变量

```
#!/bin/bash

proxy="http://127.0.0.1:8118"
function exp_proxy() {
```

(continues on next page)

(continued from previous page)

```
        export http_proxy=$proxy
        export https_proxy=$proxy
        export ftp_proxy=$proxy
        export no_proxy=localhost
}

case $1 in
        set) exp_proxy;;
        unset) unset http_proxy https_proxy ftp_proxy no_proxy;;
        *) return 0
esac
```

首先我们来了解下source的特性：即source是在当前shell而非子shell执行指定脚本中的代码

当进入bash

- 需要设置环境变量时：使用source proxy.sh set即可
- 需要取消环境变量时：使用source proxy.sh unset即可

此时如果不清楚该脚本的用途或者一时手快直接输入source proxy.sh，就可以区分exit和return

- 如果上述脚本是return 0，那么表示直接退出脚本而已，不会退出bash
- 如果上述脚本是exit 0，则表示退出当前bash，因为source是在当前shell而非子shell执行指定脚本中的代码

可能你想象不出在source执行中的return有何用处：从source来考虑，它除了用在某些脚本中加载其他环境，更主要的是在bash环境初始化脚本中使用，例如/etc/profile、~/.bashrc等，如果你在/etc/profile中用exit来替代function外面的return，那么永远也登陆不上bash

0x03 示例代码

- 随机生成密码

```
#!/bin/bash

genpasswd(){
        local l=$1
        [ "$1" == ""  ]&& l=20
        tr -dc A-Za-z0-9_</dev/urandom | head -c ${l} | xargs
}

genpasswd $1    # 将脚本传入的位置参数传递给函数，表示生成的随机密码的位数
```

- 写一个脚本，完成如下功能：
  - 1、脚本使用格式：mkscript.sh [-D|--description "script description"] [-A|--author "script author"] /path/to/somefile
  - 2、如果文件事先不存在，则创建；且前几行内容如下所示：
    - #!/bin/bash
    - # Description: script description
    - # Author: script author
  - 3、如果事先存在，但不空，且第一行不是#!/bin/bash，则提示错误并退出；如果第一行是#!/bin/bash，则使用vim打开脚本；把光标直接定位至最后一行
  - 4、打开脚本后关闭时判断脚本是否有语法错误；如果有，提示输入y继续编辑，输入n放弃并退出；如果没有，则给此文件以执行权限

```
#!/bin/bash
read -p "Enter a file: " filename
declare authname
declare descr

options(){
if [[ $# -ge 0 ]];then
        case $1 in
    -D|--description)
        authname=$4
        descr=$2
        ;;
    -A|--author)
        descr=$4
        authname=$2
        ;;
        esac
fi
}

command(){
if  bash -n $filename &> /dev/null;then
        chmod +x $filename
else
    while true;do
        read -p "[y|n]:" option
        case $option in
        y)
                vim + $filename
                ;;
        n)
                exit 8
                ;;
        esac
        done
fi
exit 6
}

oneline(){
if [[ -f $filename ]];then
        if [ `head -1 $filename` == "#!/bin/bash" ];then
        vim + $filename
        else
        echo "wrong..."
        exit 4
        fi
else
        touch $filename && echo -e "#!/bin/bash\n# Description: $descr\n# Author:
↪$authname" > $filename
        vim + $filename
fi
command
}

options $*
oneline
```

- 写一个脚本，完成如下功能：
  - 1、提示用户输入一个可执行命令
  - 2、获取这个命令所依赖的所有库文件(使用ldd命令)

  - 3、复制命令至/mnt/sysroot/对应的目录中；如果复制的是cat命令，其可执行程序的路径是/bin/cat，那么就要将/bin/cat复制到/mnt/sysroot/bin/目录中，如果复制的是useradd命令，而useradd的可执行文件路径为/usr/sbin/useradd，那么就要将其复制到/mnt/sysroot/usr/sbin/目录中
  - 4、复制各库文件至/mnt/sysroot/对应的目录中

```
#!/bin/bash

options(){
        for i in $*;do
                dirname=`dirname $i`
                [ -d /mnt/sysroot$dirname ] || mkdir -p /mnt/sysroot$dirname
                [ -f /mnt/sysroot$i ]||cp $i /mnt/sysroot$dirname/
        done
}

while true;do
        read -p "Enter a command : " pidname
        [[ "$pidname" == "quit" ]] && echo "Quit " && exit 0
        bash=`which --skip-alias $pidname`
        if [[ -x $bash ]];then
                options `/usr/bin/ldd $bash |grep -o "/[^[:space:]]\{1,\}"`
                options $bash
        else
                echo "PLZ a command!"
        fi
done

# 说明
# 将bash命令的相关bin文件和lib文件复制到/mnt/sysroot/目录中后
# 使用chroot命令可切换根目录，切换到/mnt/sysroot/后可当做bash执行复制到该处的命令，作为bash中的bash
```

- 写一个脚本，用来判定172.16.0.0网络内有哪些主机在线，在线的用绿色显示，不在线的用红色显示

```
#!/bin/bash
Cnetping(){
        for i in {1..254};do
        ping -c 1 -w 1 $1.$i
        if [[ $? -eq 0 ]];then
                echo -e -n "\033[32mping 172.16.$i.$j ke da !\033[0m\n"
        else
                echo -e -n "\033[31mping 172.16.$i.$j bu ke da !\033[0m \n"
        fi
        done
}

Bnetping(){
        for j in {0..255};do
                Cnetping $1.$j
        done
}

Bnetping 172.16
```

- 写一个脚本，用来判定随意输入的ip地址所在网段内有哪些主机在线，在线的用绿色显示，不在线的用红色显示

```
#!/bin/bash
Cnetping(){
```

(continues on next page)

(continued from previous page)

```
        for i in {1..254};do
        ping -c 1 -w 1 $1.$i
        if [[ $? -eq 0 ]];then
                echo -e -n "\033[32mping 172.16.$i.$j ke da !\033[0m\n"
        else
                echo -e -n "\033[31mping 172.16.$i.$j bu ke da !\033[0m \n"
        fi
        done
}

Bnetping(){
        for j in {0..255};do
                Cnetping $1.$j
        done
}

Anetping(){
        for m in {0.255};do
                Bnetping $1.$m
        done
}

netType=`echo $1 | cut -d'.' -f1`

if [ $netType -gt 0 -a $netType -le 126 ];then
        Anetping $1
elif [ $netType -ge 128 -a $netType -le 191 ];then
        Bnetping $1
elif [ $netType -ge 192 -a $netType -le 223 ];then
        Cnetping $1
else
        echo "Wrong"
        exit 3
fi
```

知识碎片

shell脚本是一种纯面向过程的脚本编程语言

在编写shell脚本时，需要注意以下几点

- 标准输出：在编写shell脚本的时候，要考虑下该命令语句是否存在标准输出
  - 如果有，是否需要输出到标准输出设备上
  - 如果不需要，那就输出重定向至/dev/null
- 常见逻辑错误
  - 用户输入是否为空问题
  - 用户输入字符串大小写问题
  - 用户输入是否存在问题
- 编程思想
  - 明确脚本的输入、输出是什么
  - 根据输入考虑可能存在逻辑错误的地方
  - 根据输出判断使用什么控制流程
  - 在保证功能实现的前提下进行优化精简代码

- 语法解析规则
  - shell中0为真，非0为假
  - 赋值操作符=右边必须是常量
    - 数值型常量：即0,1,2,.....
    - 字符串型常量：最好使用""括起来；要不然如果中间有空格，就会当做命令执行
    - 数组型常量：例如(1 'b' 3 'a')
    - 列表型常量：例如{1..4}
  - 以下情况下，字符串会被当做命令直接执行
    - =右边没有空格分隔的字符串默认是字符串常量，如果有空格分隔，则被解析成命令执行；所以此时使用字符串时，不管其中有没有空格，最后都用""操作符将其包围
    - if/elif后面的字符串如果没有被[]、[[]]条件测试操作符包围，则会被解析成命令执行

```
if 0; then
        echo true
else
        echo false
fi
# 此处0被当做命令执行，但是没有该命令，所以该执行状态返回值为假，结果打印false

if [ 0 ]; then
        echo true
else
        echo false
fi
# 此处0被[]操作符包围，表示执行测试表达式，由于在shell中0为真，所以结果打印true
```

在编写shell脚本时，常用到的一些命令语句

- 判断用户是否存在
  - grep "^$userName\>" /etc/passwd &> /dev/null
  - id $userName
- 获取用户的相关信息(用户名，UID，GID或者默认shell)
  - 对/etc/passwd文件进行处理
  - 使用id命令
- 脚本文件中导入调用其它脚本文件
  - source config_file
  - . config_file

```
#!/bin/bash
# configurefile: /tmp/script/myscript.conf

# 先判断对导入文件是否有读权限，然后尝试导入
[ -r /tmp/script/myscript.conf ] && . /tmp/script/myscript.conf
# 如果导入文件没有成功或者导入文件中对引用变量没有相关定义时，需定义默认值，防止出错
userName=${userName:-testuser}

echo $userName
```

- 读取文件内容

```
while read line; do
        CMD_LIST
done < /path/to/somefile
```

- 下载文件

```
#!/bin/bash

url="http::/mirrors.aliyun.com/centos/centos6.5.repo"

which wget &> /dev/null || exit 5 # 如果wget命令不存在就退出

downloader=`which wget`  # 获取wget命令的二进制文件路径

[ -x $downloader ] || exit 6 # 如果二进制文件没有执行权限就退出

$downloader $url
```

- 创建临时文件或目录

```
mktemp [-d] /tmp/file.XX    # X指定越多，随机生成的后缀就越长，其中-d表示创建临时目录
```

常用类库

shell脚本中常用的类库可以分为三大类

常用环境变量

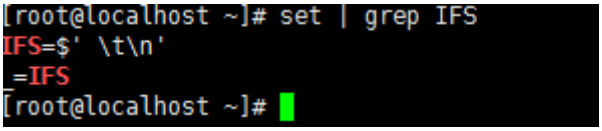
shell脚本中常用的环境变量有

- *IFS*
- *RANDOM*

0x00 IFS

shell下的很多命令都会分割单词，绝大多数时候默认是采用空格作为分隔符，有些时候遇到制表符、换行符也会进行分隔；这种分隔符是由IFS环境变量指定的

IFS是shell内部字段分隔符的环境变量

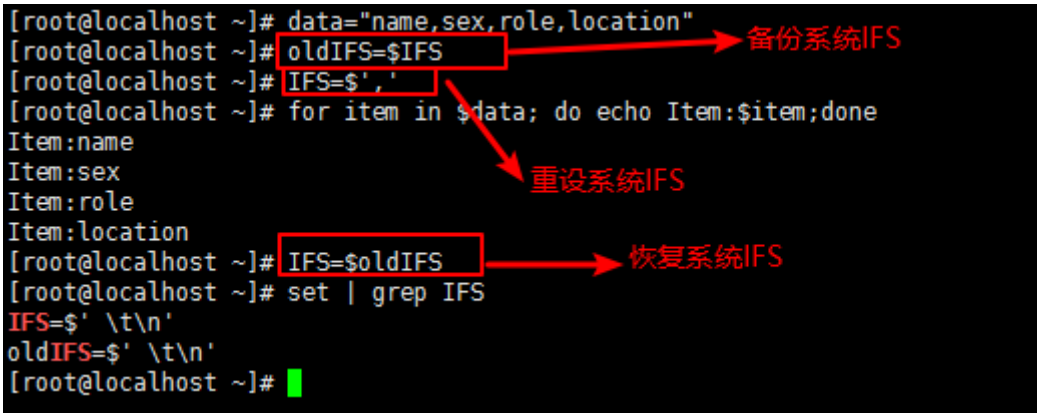


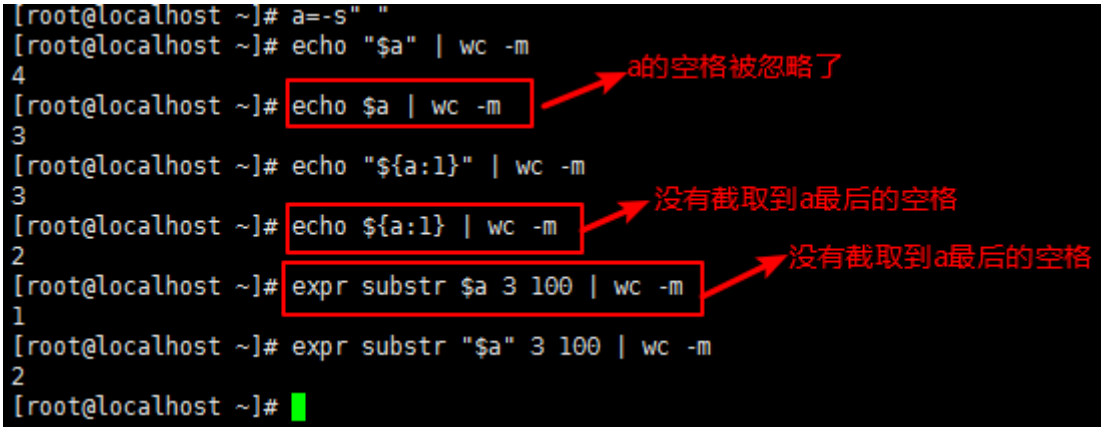
由上图可知：默认的IFS在碰到空格、制表符\t或 分行符\n就会自动分隔进入下一步；但是对空格处理有点不一样，对行首和行尾两边的空格不处理，并且多个连续的空格默认当作一个空格

有些时候在编写脚本或执行循环的时候，修改IFS可以起很大作用。如果要修改IFS，最好记得先备份系统IFS，再需要的地方再还原IFS

大多数时候，我们都不会去修改IFS来达到某种目的，而是采用其他方法来替代实现。这样就需要注意默认IFS的一个特殊性，它会忽略前导空白和后缀空白，并压缩连续空白；但是在某些时候，这会出现意想不到的问题

因此，**在可以对变量加引号的情况下，一定要加上引号来保护空白字符**





0x01 RANDOM

RANDOM环境变量是bash的伪随机数生成器

- $RANDOM=======生成0~32767之间的随机数
- $[RANDOM%num]===生成0~num之间的随机数；对算术表达式的值进行引用时需要使用[]

代码示例：通过脚本生成n个随机数(N>5),对这些随机数按从小到大排序

```
#!/bin/bash
declare -a arrynumber
read -p "Enter a number:" opt
opt=$[opt-1]
for i in `seq 0 $opt`;do
  arrynumer[$i]=$[RANDOM%1000]
done
let length=${#arrynumer[@]}
length=$[length-1]
for i in `seq 0 $length`;do
  let j=i+1
  for j in `seq $j $length`; do
      if [ ${arrynumer[$j]} -lt ${arrynumer[$i]} ];then
        temp=${arrynumer[$j]}
        arrynumer[$j]=${arrynumer[$i]}
        arrynumer[$i]=$temp
      fi
  done
  echo ${arrynumer[$i]}
done
```

常用命令

shell脚本可以理解为shell命令的集合，shell命令可以分为两大类

- `shell内部命令(builtin command)`：在bash中内部实现的命令叫做内建命令，在文件系统上没有对应的可执行文件
- `shell外部命令(binary command)`：在文件系统上的某个位置(/bin、/sbin等)有一个与命令名称对应的可执行文件

关于shell脚本中可能会用到的shell命令可以参考：linux工具集之shell命令

在此处我们主要介绍下shell脚本中使用频次最高的几个命令

- *read*：获取用户输入
- *echo*：打印输出
- *printf*：打印输出
- *shift*：剔除位置参数

0x00 read

关于`read`命令的用法详见：IO操作之read

0x01 echo

关于`echo`命令的用法详见：IO操作之echo

0x02 printf

关于`printf`命令的用法详见：IO操作之printf

0x03 shift

`shift`命令在shell脚本中用来剔除脚本的位置变量，它是shell内建命令，使用`help shift`命令可以查看其语法格式和使用说明，它的语法格式如下

```
shift [n]
```

该命令常用来解析脚本的传入参数

- `shift`表示剔除脚本的第一个传入参数，后面参数往前排
- `shift n`表示剔除脚本的前n个传入参数，后面参数往前排

写一个脚本，使用形式：`userinfo.sh -u username [-v {1|2}]`

- `-u`选项用于指定用户，而后脚本显示用户的UID和GID
- `-v`选项后面是1，则显示用户的家目录路径；如果是2，则显示用户的家目录路径和shell

```
#!/bin/bash

[ $# -lt 2 ] && echo "less arguments" && exit 3

if [[ "$1" == "-u" ]]; then
        userName="$2"
        shift 2         # 剔除前2个位置参数
```

(continues on next page)

(continued from previous page)

```
fi

if [[ $# -ge 2 ]] && [ "$1" == "-v" ]; then
        verFlag=$2
fi

verFlag=${verFlag:-0}

if [ -n $verFlag ]; then
        if ! [[ $verFlag =~ [012] ]]; then
                echo "Wrong Parameter"
                echo "Usage: `basename $0` -u UserName -v {1|2}"
                exit 4
        fi
fi

if [ $verFlag -eq 1 ];then
        grep "^$userName" /etc/passwd | cut -d: -f1,3,4,6
elif [ $verFlag -eq 2 ];then
        grep "^$userName" /etc/passwd | cut -d: -f1,3,4,6,7
else
        grep "^$userName" /etc/passwd | cut -d: -f1,3,4
fi
```

常用函数

参考文档：functions文件详细分析和说明

shell中函数和命令不一样，它没有对应的二进制文件，只有相关的声明定义

shell中函数可以大致分为两大类：自定义函数和库函数

自定义函数好说，直接在脚本中自行声明定义和调用即可；在这里我们主要是介绍库函数，shell中所谓库函数就是`/etc/rc.d/init.d/functions`文件中定义的系统函数，这些系统函数几乎被`/etc/rc.d/init.d/`下所有的sysv服务启动脚本加载，在该文件中提供了以下几个非常有用的函数

- 显示函数
  - *success*：显示绿色的`OK`，表示成功
  - *failure*：显示红色的`FAILED`，表示失败
  - *passed*：显示黄色的`PASSED`，表示pass该任务
  - *warning*：显示黄色的`warning`，表示警告
  - *confirm*：提示`(Y)es/(N)o/(C)ontinue? [Y]`并判断、传递输入的值
  - *is_true*：`$1`的布尔值代表为真时，返回状态码`0`，否则返回`1`；包括`t/y/yes/true`，不区分大小写
  - *is_false*：`$1`的布尔值代表为假时，返回状态码`0`，否则返回`1`；包括`f/n/no/false`，不区分大小写
  - *action*：根据进程退出状态码自行判断是执行`success`还是`failure`
- 进程函数
  - *checkpid*：检查`/proc`下是否有给定`pid`对应的目录，给定多个`pid`时，只要存在一个目录都返回状态码`0`
  - *__pids_var_run*：检查`pid`是否存在，并保存到变量`pid`中，同时返回几种进程状态码
  - *__pids_pidof*：获取进程`pid`
  - *pidfileofproc*：获取进程`pid`，但只能获取`/var/run`下的`pid`文件中的值

  - *pidofproc*：获取进程pid，可获取任意给定pidfile或默认/var/run下pidfile中的值
  - *status*：检查给定进程的运行状态
  - *daemon*：启动一个服务程序，启动前还检查进程是否已在运行
  - *killproc*：杀掉给定的服务进程

以下是/etc/init.d/functions文件的开头定义的语句(本文分析的/etc/init.d/functions文件是CentOS 7上的，和CentOS 6有些许区别)

- 设置umask值，使得加载该文件的脚本所在shell的umask为22
- 导出PATH路径变量，但这个导出的路径变量并不理想，因为要为非rpm包安装的程序设计服务启动脚本时，必须写全路径命令，例如/usr/local/mysql/bin/mysql，因此，可以考虑将/etc/init.d/functions中的该语句注释掉

```
# Make sure umask is sane
umask 022

# Set up a default search path.
PATH="/sbin:/usr/sbin:/bin:/usr/bin"
export PATH
```

0x00 显示函数

显示函数常用在编写系统服务启动脚本，便于提示相关启动信息

0x0000 success

除了success函数，还有echo_success函数也可以显示绿色的OK，表示成功

以下是echo_success和success函数的定义语句

```
echo_success() {
    [ "$BOOTUP" = "color" ] && $MOVE_TO_COL
    echo -n "["
    [ "$BOOTUP" = "color" ] && $SETCOLOR_SUCCESS
    echo -n $"  OK  "
    [ "$BOOTUP" = "color" ] && $SETCOLOR_NORMAL
    echo -n "]"
    echo -ne "\r"
    return 0
}


success() {
        [ "$BOOTUP" != "verbose" -a -z "${LSB:-}" ] && echo_success
        return 0
}
```

这两个函数的功能就是：不换行带绿色输出[OK]字样；效果如下

```
[root@localhost init.d]# . /etc/init.d/functions
[root@localhost init.d]# success
[root@localhost init.d]#                                   [  OK  ]
[root@localhost init.d]# echo_success
[root@localhost init.d]#                                   [  OK  ]
```

0x0001 failure

除了failure函数，还有echo_failure函数也可以显示红色的FAILED，表示失败

以下是echo_failure和failure函数的定义语句

```
echo_failure() {
    [ "$BOOTUP" = "color" ] && $MOVE_TO_COL
    echo -n "["
    [ "$BOOTUP" = "color" ] && $SETCOLOR_FAILURE
    echo -n $"FAILED"
    [ "$BOOTUP" = "color" ] && $SETCOLOR_NORMAL
    echo -n "]"
    echo -ne "\r"
    return 1
}


failure() {
    local rc=$?
    [ "$BOOTUP" != "verbose" -a -z "${LSB:-}" ] && echo_failure
    [ -x /bin/plymouth ] && /bin/plymouth --details
    return $rc
}
```

这两个函数的功能就是：不换行带红色输出[FAILED]字样；效果如下

```
[root@localhost init.d]# . /etc/init.d/functions
[root@localhost init.d]# failure
[root@localhost init.d]#                                   [FAILED]
[root@localhost init.d]# echo_failure
[root@localhost init.d]#                                   [FAILED]
```

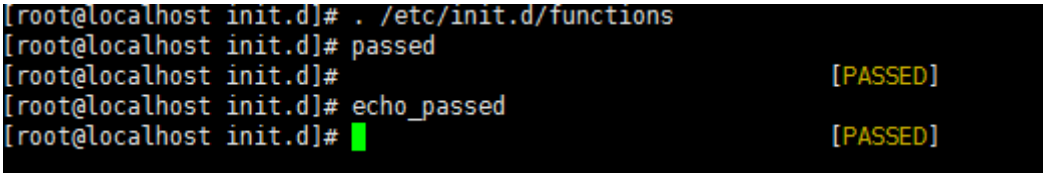
0x0002 passed

除了passed函数，还有echo_passed函数也可以显示黄色的PASSED，表示pass该任务

以下是echo_passed和passed函数的定义语句

```
echo_passed() {
    [ "$BOOTUP" = "color" ] && $MOVE_TO_COL
    echo -n "["
    [ "$BOOTUP" = "color" ] && $SETCOLOR_WARNING
    echo -n $"PASSED"
    [ "$BOOTUP" = "color" ] && $SETCOLOR_NORMAL
    echo -n "]"
    echo -ne "\r"
    return 1
}


passed() {
    local rc=$?
    [ "$BOOTUP" != "verbose" -a -z "${LSB:-}" ] && echo_passed
    return $rc
}
```

这两个函数的功能就是：不换行带黄色输出[PASSED]字样；效果如下



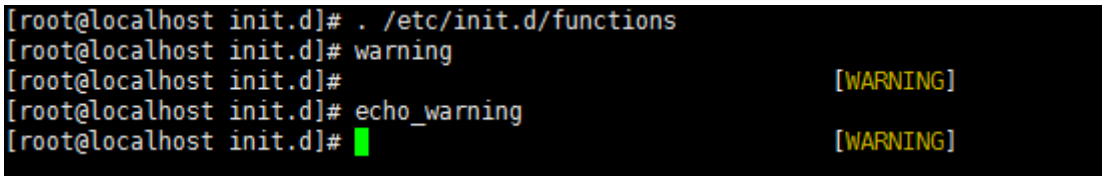
0x0003 warning

除了warning函数，还有echo_warning函数也可以显示黄色的warning，表示警告

以下是echo_warning和warning函数的定义语句

```
echo_warning() {
    [ "$BOOTUP" = "color" ] && $MOVE_TO_COL
    echo -n "["
    [ "$BOOTUP" = "color" ] && $SETCOLOR_WARNING
    echo -n $"WARNING"
    [ "$BOOTUP" = "color" ] && $SETCOLOR_NORMAL
    echo -n "]"
    echo -ne "\r"
    return 1
}


warning() {
    local rc=$?
    [ "$BOOTUP" != "verbose" -a -z "${LSB:-}" ] && echo_warning
    return $rc
}
```

这两个函数的功能就是：不换行带黄色输出[WARNING]字样；效果如下



0x0004 confirm

这个函数一般用不上，因为脚本本来就是为了避免交互式的。在CentOS 7的functions中已经删除了该函数定义语句。不过，借鉴下它的处理方法还是不错的

以下摘自CentOS 6.6的/etc/init.d/functions文件

```
# returns OK if $1 contains $2
strstr() {
        [ "${1#*$2*}" = "$1" ] && return 1    # 参数$1中不包含$2时，返回1，否则返回0
        return 0
}

# Confirm whether we really want to run this service
confirm() {
        [ -x /bin/plymouth ] && /bin/plymouth --hide-splash
        while : ; do
                echo -n $"Start service $1 (Y)es/(N)o/(C)ontinue? [Y] "
                read answer
                if strstr $"yY" "$answer" || [ "$answer" = "" ] ; then
```

(continues on next page)

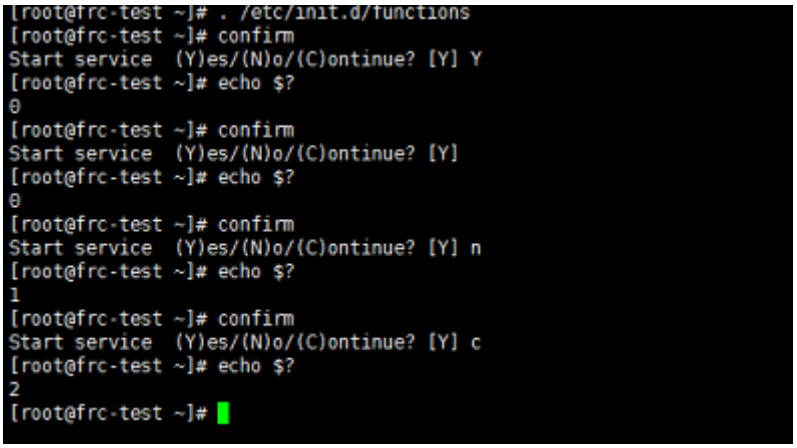
(continued from previous page)

```
                return 0
        elif strstr $"cC" "$answer" ; then
                rm -f /var/run/confirm
                [ -x /bin/plymouth ] && /bin/plymouth --show-splash
                return 2
        elif strstr $"nN" "$answer" ; then
                return 1
        fi
        done
}
```

上述代码中

- 第一个函数strstr的作用是判断第一个参数$1中是否包含了$2，如果包含了则返回状态码0,，这函数也是一个不错的技巧
- 第二个函数confirm的作用是根据交互式输入的值返回不同的状态码，如果输入的是y或Y或不输入时，返回0。输入的是c或C时，返回状态码2，输入的是n或“N“时返回状态码1

于是可以根据confirm的状态值决定是否要继续执行某个程序，用法和效果如下



0x0005 is_true

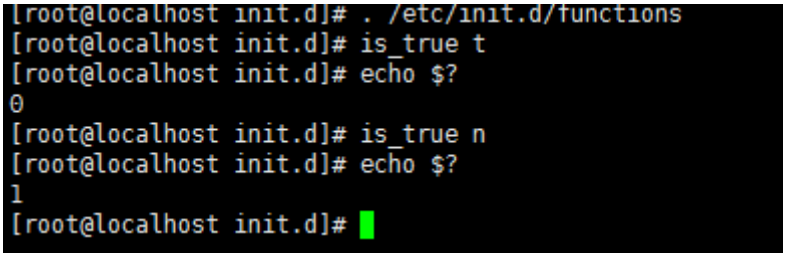
以下是is_true函数的定义语句

```
# Evaluate shvar-style booleans
is_true() {
    case "$1" in
    [tT] | [yY] | [yY][eE][sS] | [oO][nN] | [tT][rR][uU][eE] | 1)
        return 0
        ;;
    esac
    return 1
}
```

由以上代码可知：这个函数的作用就是转换输入的布尔值为状态码；$1第一个函数参数的布尔值代表为真(包括t/y/yes/true，不区分大小写)时，返回状态码0，否则返回1

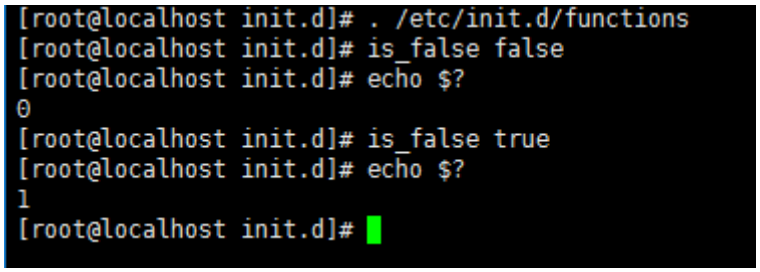
0x0006 is_false

以下是is_false函数的定义语句




```
# Evaluate shvar-style booleans
is_false() {
    case "$1" in
    [fF] | [nN] | [nN][oO] | [oO][fF][fF] | [fF][aA][lL][sS][eE] | 0)
        return 0
        ;;
    esac
    return 1
}
```

由以上代码可知：这个函数的作用就是转换输入的布尔值为状态码；$1第一个函数参数的布尔值代表为假(包括f/n/no/false，不区分大小写)时，返回状态码0，否则返回1



0x0007 action

该函数在写脚本时还比较有用，可以根据退出状态码自动判断是执行success还是执行failure函数

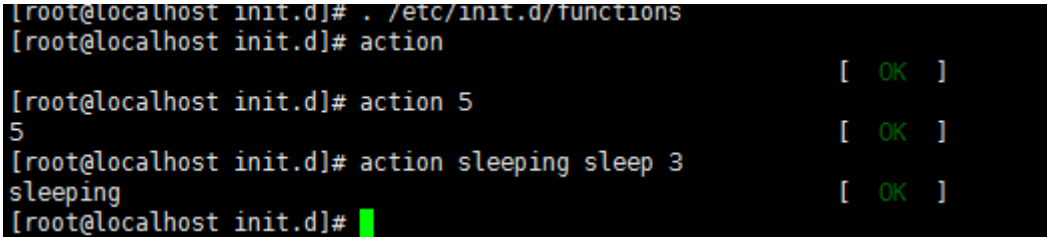
以下是action函数的定义语句

```
# Run some action. Log its output.
action() {
    local STRING rc

    STRING=$1
    echo -n "$STRING "
    shift
    "$@" && success $"$STRING" || failure $"$STRING"
    rc=$?
    echo
    return $rc
}
```

这个函数定义的很有技巧

- 先将第一个参数保存并踢掉，再执行后面的命令("$@"表示执行后面的命令)
- 当action函数只有一个参数时，action直接返回OK，状态码为0；当超过一个参数时，第一个参数先被打印，再执行从第二个参数开始的命令

在脚本中使用action函数时，可以让命令执行成功与否的判断显得更专业，效果如下



通常，该函数会结合/bin/true和/bin/false命令使用，它们无条件返回0或1状态码；例如，mysqld启动脚本中，判断mysqld已在运行时，直接输出启动ok的消息，但实际上根本没做任何事

```
# action函数使用格式
# action $"MESSAGES: " /bin/true
# action $"MESSAGES: " /bin/false

if [ $MYSQLDRUNNING = 1 ] && [ $? = 0 ]; then
        # already running, do nothing
        action $"Starting $prog: " /bin/true
        ret=0
fi
```

0x01 进程函数

启动进程时，pid文件非常重要

- pid文件不仅可以用来判断进程是否在运行，还可以从中读取pid号用来杀进程
- pid文件中可能有多行，表示多实例

0x0100 checkpid

checkpid函数是用来检测给定的`pid值在/proc下是否有对应的目录存在

以下是函数checkpid的定义语句

```
# Check if any of $pid (could be plural) are running
checkpid() {
    local i

    for i in $* ; do       # 检测/proc目录下是否存在给定的进程目录
        [ -d "/proc/$i" ] && return 0
    done
    return 1
}
```

每个进程都必有一个pid，但并不一定都记录在pid文件中，例如线程的pid；但无论如何，在/proc/目录下，一定会有pid号命名的目录，只要有对应pid号的目录，就表示该进程已经在运行

在检查/proc下是否有给定pid对应的目录，无论给定多少个pid，只要有一个有目录，都返回0

该函数的调用方法如下

```
checkpid pid_list
```

效果图如下

```
[root@localhost ~]# source /etc/init.d/functions
[root@localhost ~]# sleep 10 & a="$!";sleep 10 & a="$a $!";sleep 10 & a="$a $!";checkpid $a
[1] 7882
[2] 7883
[3] 7884
[root@localhost ~]# echo $?
0
[root@localhost ~]#
```

0x0101 __pids_var_run

__pids_var_run函数是用来判断给定程序的运行状态以及对应的pid文件是否存在

以下是函数__pids_var_run的定义语句

```
# __proc_pids {program} [pidfile]
# Set $pid to pids from /var/run* for {program}.  $pid should be declared
# local in the caller.
# Returns LSB exit code for the 'status' action

# 通过检测pid判断程序是否已在运行
__pids_var_run() {
    local base=${1##*/}                         # 获取进程名的basename
    local pid_file=${2:-/var/run/$base.pid}     # 定义pid文件路径
    local pid_dir=$(/usr/bin/dirname $pid_file > /dev/null)
    local binary=$3

    [ -d "$pid_dir" -a ! -r "$pid_dir" ] && return 4

    pid=
    if [ -f "$pid_file" ] ; then    # 判断给定的pid文件是否存在
            local line p

        [ ! -r "$pid_file" ] && return 4 # "user had insufficient privilege"
        while : ; do                        # 将pid文件中的pid值赋值给pid变量
            read line
            [ -z "$line" ] && break
            for p in $line ; do
                if [ -z "${p//[0-9]/}" ] && [ -d "/proc/$p" ] ; then
                    if [ -n "$binary" ] ; then
                        local b=$(readlink /proc/$p/exe | sed -e 's/\s*(deleted)$//
↪')
                        [ "$b" != "$binary" ] && continue
                    fi
                    pid="$pid $p"
                fi
            done
        done < "$pid_file"

            if [ -n "$pid" ]; then # pid存在，则返回0，否则表示pid文件存在，但/proc下没有
对应命令
                    return 0        # 即进程已死，但pid文件却存在，返回状态码1
            fi
        return 1 # "Program is dead and /var/run pid file exists"
    fi
    return 3 # "Program is not running"pid文件不存在时，表示进程未进行，返回状态码3
}
```

由函数定义可知：只有当pid文件存在，且/proc下有pid对应的目录时，才表示进程在运行(当然线程没有pid文件)，该函数的调用方法是：__pids_var_run program [pidfile]

- program为程序进程名

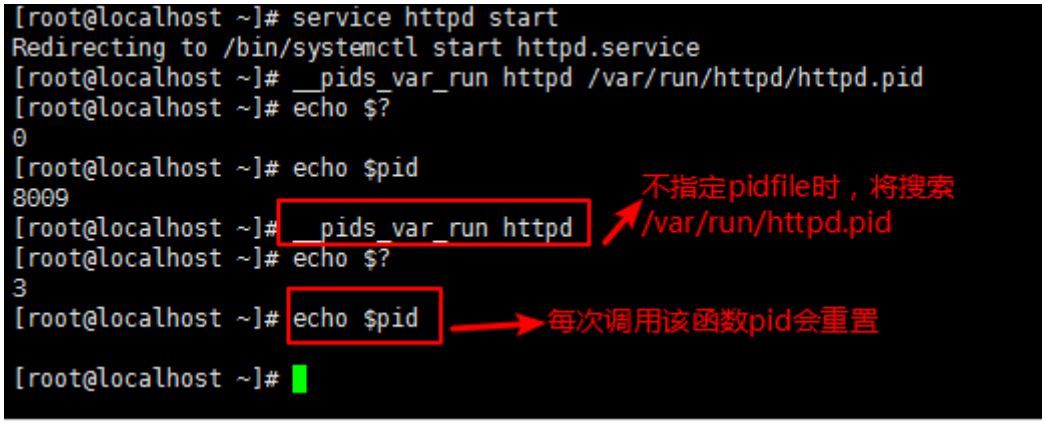
- pidfile为进程pid文件名，如果不给定pidfile，则默认为/var/run/$base.pid文件
  - pidfile的路径可能为/var/run/$base.pid文件($base表示进程名的basename)，此路径为默认值
  - pidfile的路径也可能是自定义的路径，例如mysql的pid可以自定义为/mysql/data/mysql01.pid
- 函数的执行结果有4种状态返回码
  - 0：表示program正在运行
  - 1：表示program进程已死，pid文件存在，但/proc目录下没有对应的文件
  - 3：表示pid文件不存在
  - 4：表示pid文件的权限错误，不可读
- 函数还会保存变量pid的结果，以供其他程序引用

这个函数非常重要，不仅可以从pidfile中获取并保存pid号码，还根据情况返回几种状态码，这几个状态码是status函数的重要依据，在SysV服务启动脚本中使用非常广泛

该函数的调用方法如下

```
__pids_var_run program [pidfile]
```

以下是httpd进程的测试结果，分别是指定pid文件和不指定pid文件的情况



0x0102 __pids_pidof

__pids_pidof函数是用来获取给定进程的pid

以下是函数__pids_pidof的定义语句

```
# Output PIDs of matching processes, found using pidof
# 忽略当前shell的PID，父shell的PID，调用pidof程序shell的PID
__pids_pidof() {
    pidof -c -m -o $$ -o $PPID -o %PPID -x "$1" || \
        pidof -c -m -o $$ -o $PPID -o %PPID -x "${1##*/}"
}
```

由以上代码可知：该函数使用了pidof命令，获取给定进程的pid值会更加精确，其中使用了几个-o选项，它用于忽略指定的pid

- -o $$表示忽略当前shell进程PID，大多数时候它会继承父shell的pid，但在脚本中时它代表的是脚本所在shell的pid
- -o $PPID表示忽略父shell进程PID
- -o %PPID表示忽略调用pidof命令的shell进程PID

关于pidof命令我们在这里简单介绍下，示例脚本如下

```
#!/bin/bash

echo 'pidof bash: '`pidof bash`
echo 'script shell pid: '`echo $$`
echo 'script parent shell pid: '`echo $PPID`
echo 'pidof -o $$ bash: '`pidof -o $$ bash`
echo 'pidof -o $PPID bash: '`pidof -o $PPID bash`
echo 'pidof -o %PPID bash: '`pidof -o %PPID bash`
echo 'pidof -o $$ -o $PPID -o %PPID bash: '`pidof -o $$ -o $PPID -o %PPID bash`
```

效果如下

```
[root@localhost ~]# pidof bash
7306 6942 1552
[root@localhost ~]# (echo 'parent shell: '$$;echo "current bash pid: `pidof bash`";./test.sh)|cat -n
     1  parent shell: 6942
     2  current bash pid: 7337 7306 6942 1552
     3  pidof bash: 7340 7337 7306 6942 1552
     4  script shell pid: 7340
     5  script parent shell pid: 7337
     6  pidof -o $$ bash: 7337 7306 6942 1552
     7  pidof -o $PPID bash: 7340 7306 6942 1552
     8  pidof -o %PPID bash: 7337 7306 6942 1552
     9  pidof -o $$ -o $PPID -o %PPID bash: 7306 6942 1552
[root@localhost ~]#
```

上述效果图中

- 第一个pidof命令显示结果中说明当前已有3个bash，pid分别为3306、2436、2302
- 第二个命令显示结果中
  - 行1说明括号的父shell为6942
  - 行5说明脚本的父shell为7337。即括号的父shell为当前bash环境，脚本的父shell为括号所在shell
  - 行2减第一个命令的结果说明括号所在子shell的pid为7337
  - 行3减行2说明shell脚本所在子shell的pid为7340
  - -o $$忽略的是当前shell，即脚本所在shell的pid，因为在shell脚本中时，$$不继承父shell的pid
  - -o $PPID忽略的是pidof所在父shell，即括号所在shell
  - -o %PPID忽略的是调用pidof程序所在的shell，即脚本所在shell

0x0103 pidfileofproc

pidfileofproc函数用来获取给定程序的pid，注意该函数不是获取pidfile，而是获取pid值

以下是函数pidfileofproc的定义语句

```
# A function to find the pid of a program. Looks *only* at the pidfile
pidfileofproc() {
    local pid

    # Test syntax.
    if [ "$#" = 0 ] ; then
        echo $"Usage: pidfileofproc {program}"
        return 1
    fi

    __pids_var_run "$1"          # 不提供pidfile，因此认为是/var/run/$base.pid
```

(continues on next page)

(continued from previous page)

```
    [ -n "$pid" ] && echo $pid
    return 0
}
```

由以上代码可知：pidfileofproc函数只能获取/var/run下的pid值

该函数用的比较少，但确实有使用它的脚本；如crond启动脚本中借助pidfileofproc来杀进程

```
echo -n $"Stopping $prog: "
if [ -n "`pidfileofproc $exec`" ]; then
        killproc $exec
        RETVAL=3
else
        failure $"Stopping $prog"
fi
```

0x0104 pidofproc

pidofproc函数也可以用来获取给定程序的pid，注意该函数不是获取pidfile，而是获取pid值

以下是函数pidofproc的定义语句

```
# A function to find the pid of a program.
pidofproc() {
    local RC pid pid_file=

    # Test syntax.
    if [ "$#" = 0 ]; then
        echo $"Usage: pidofproc [-p pidfile] {program}"
        return 1
    fi
    if [ "$1" = "-p" ]; then     # 既可以获取/var/run/$base.pid中的pid
        pid_file=$2              # 也可以获取自给定pid文件中的pid
        shift 2
    fi
    fail_code=3 # "Program is not running"

    # First try "/var/run/*.pid" files
    __pids_var_run "$1" "$pid_file"
    RC=$?
    if [ -n "$pid" ]; then       # $pid不为空时，输出program的pid值
        echo $pid
        return 0
    fi

    [ -n "$pid_file" ] && return $RC   #  $pid为空，但使用了"-p"指定pidfile时，返回$RC
    __pids_pidof "$1" || return $RC    #  $pid为空，且$pidfile为空时，获取进程号pid并输出
}
```

由以上代码可知：pidofproc函数既可以获取/var/run下的pid值，又可以获取自给定pidfile中的pid值

该函数用的比较少，但确实有使用它的脚本；如dnsbind的named服务启动脚本中借助pidofproc来判断进程是否已在运行

```
pidofnamed() {
        pidofproc -p "$ROOTDIR$PIDFILE" "$named";
}

if [ -n "`pidofnamed`" ]; then
```

(continues on next page)

(continued from previous page)

```
        echo -n $"named: already running"
        success
        echo
        exit 0
fi
```

0x0105 daemon

daemon函数用于启动一个程序，并根据结果输出success或failure

daemon函数的定义语句如下

```
# A function to start a program.
daemon() {
    # Test syntax.
    local gotbase= force= nicelevel corelimit
    local pid base= user= nice= bg= pid_file=
    local cgroup=
    nicelevel=0
    while [ "$1" != "${1##[-+]}" ]; do
        case $1 in
        '')
            echo $"$0: Usage: daemon [+/-nicelevel] {program}" "[arg1]..."
            return 1
            ;;
        --check)
            base=$2
            gotbase="yes"
            shift 2
            ;;
        --check=?*)
            base=${1#--check=}
            gotbase="yes"
            shift
            ;;
        --user)
            user=$2
            shift 2
            ;;
        --user=?*)
            user=${1#--user=}
            shift
            ;;
        --pidfile)
            pid_file=$2
            shift 2
            ;;
        --pidfile=?*)
            pid_file=${1#--pidfile=}
            shift
            ;;
        --force)
            force="force"
            shift
            ;;
        [-+][0-9]*)
            nice="nice -n $1"
            shift
            ;;
        *)
```

(continues on next page)

(continued from previous page)

```
            echo $"$0: Usage: daemon [+/-nicelevel] {program}" "[arg1]..."
            return 1
            ;;
      esac
    done

    # Save basename.
    [ -z "$gotbase" ] && base=${1##*/}

    # See if it's already running. Look *only* at the pid file.
    __pids_var_run "$base" "$pid_file"

    [ -n "$pid" -a -z "$force" ] && return

    # make sure it doesn't core dump anywhere unless requested
    corelimit="ulimit -S -c ${DAEMON_COREFILE_LIMIT:-0}"

    # if they set NICELEVEL in /etc/sysconfig/foo, honor it
    [ -n "${NICELEVEL:-}" ] && nice="nice -n $NICELEVEL"

    # if they set CGROUP_DAEMON in /etc/sysconfig/foo, honor it
    if [ -n "${CGROUP_DAEMON}" ]; then
        if [ ! -x /bin/cgexec ]; then
            echo -n "Cgroups not installed"; warning
            echo
        else
            cgroup="/bin/cgexec";
            for i in $CGROUP_DAEMON; do
                cgroup="$cgroup -g $i";
            done
        fi
    fi

    # Echo daemon
    [ "${BOOTUP:-}" = "verbose" -a -z "${LSB:-}" ] && echo -n " $base"

    # And start it up.
    if [ -z "$user" ]; then
       $cgroup $nice /bin/bash -c "$corelimit >/dev/null 2>&1 ; $*"
    else
       $cgroup $nice runuser -s /bin/bash $user -c "$corelimit >/dev/null 2>&1 ; $*
↪"
    fi

    [ "$?" -eq 0 ] && success $"$base startup" || failure $"$base startup"
}
```

daemon函数调用方法

```
daemon [--check=servicename] [--user=USER] [--pidfile=PIDFILE] [--force] program␣
↪[prog_args]
```

其中需要注意的是

- 只有--user选项可以用来控制program启动的环境
- --check和--pidfile选项都是用来检查是否已运行的，不是用来启动的，如果提供了--check，则检查的是名为`servicename的进程，否则检查的是program名称的进程
- --force则表示进程已存在时仍启动
- prog_args是向program传递它的运行参数，一般会从/etc/sysconfig/$base文件中获取

例如httpd的启动脚本中

```
echo -n $"Starting $prog: "
daemon --pidfile=${pidfile} $httpd $OPTIONS
```

其执行结果大致如下

```
[root@xuexi ~]# service httpd start
Starting httpd:                                [  OK  ]
```

还需注意，通常program的运行参数可能也是--开头的，要和program前面的选项区分。例如

```
daemon --pidfile $pidfile --check $servicename $processname --pid-file=$pidfile
```

其中

- 第二个--pid-file是$processname的运行参数
- 第一个--pidfile是daemon检测$processname是否已运行的选项
- 由于提供了--check $servicename，所以函数调用语句__pids_var_run $base [pidfile]中的$base等于$servicename，即表示检查$servicename进程是否允许；如果没有提供该选项，则检查的是$processname

在SysV脚本中，daemon会配合以下几个语句同时执行

```
echo -n $"Starting $prog: "
daemon --pidfile=${pidfile} $prog $OPTIONS
RETVAL=$?
[ $RETVAL = 0 ] && touch ${lockfile}
return $RETVAL
```

daemon函数启动程序时，自身就会调用success或failure函数，所以就不需再使用action函数了；如果不使用daemon函数启动服务，通常会配合action函数，例如：

```
$prog $OPTIONS
RETVAL=$?
[ $RETVAL -eq 0 ] && action "Starting $prog" /bin/true && touch ${lockfile}
```

0x0106 killproc

killproc函数的作用是根据给定程序名杀进程；中间它会获取程序名对应的pid号，且保证/proc目录下没有pid对应的目录才表示进程关闭成功

killproc函数的定义语句如下

```
# A function to stop a program.
killproc() {
    local RC killlevel= base pid pid_file= delay try binary=

    RC=0; delay=3; try=0
    # Test syntax.
    if [ "$#" -eq 0 ]; then
        echo $"Usage: killproc [-p pidfile] [ -d delay] {program} [-signal]"
        return 1
    fi
    if [ "$1" = "-p" ]; then
        pid_file=$2
        shift 2
    fi
    if [ "$1" = "-b" ]; then
        if [ -z $pid_file ]; then
            echo $"-b option can be used only with -p"
```

(continues on next page)

(continued from previous page)

```
            echo $"Usage: killproc -p pidfile -b binary program"
            return 1
        fi
        binary=$2
        shift 2
    fi
    if [ "$1" = "-d" ]; then
        if [ "$?" -eq 1 ]; then
            echo $"Usage: killproc [-p pidfile] [ -d delay] {program} [-signal]"
            return 1
        fi
        shift 2
    fi


    # check for second arg to be kill level
    [ -n "${2:-}" ] && killlevel=$2

    # Save basename.
    base=${1##*/}

    # Find pid.
    __pids_var_run "$1" "$pid_file" "$binary"
    RC=$?
    if [ -z "$pid" ]; then
        if [ -z "$pid_file" ]; then
            pid="$(__pids_pidof "$1")"
        else
            [ "$RC" = "4" ] && { failure $"$base shutdown" ; return $RC ;}
        fi
    fi

    # Kill it.
    if [ -n "$pid" ] ; then
        [ "$BOOTUP" = "verbose" -a -z "${LSB:-}" ] && echo -n "$base "
        if [ -z "$killlevel" ] ; then
            __kill_pids_term_kill -d $delay $pid
            RC=$?
            [ "$RC" -eq 0 ] && success $"$base shutdown" || failure $"$base␣
↪shutdown"
        # use specified level only
        else
            if checkpid $pid; then
                kill $killlevel $pid >/dev/null 2>&1
                RC=$?
                [ "$RC" -eq 0 ] && success $"$base $killlevel" || failure $"$base
↪$killlevel"
            elif [ -n "${LSB:-}" ]; then
                RC=7 # Program is not running
            fi
        fi
    else
        if [ -n "${LSB:-}" -a -n "$killlevel" ]; then
            RC=7 # Program is not running
        else
            failure $"$base shutdown"
            RC=0
            __kill_pids_term_kill -d $delay $pid
            RC=$?
            [ "$RC" -eq 0 ] && success $"$base shutdown" || failure $"$base␣
↪shutdown"
```

(continues on next page)

(continued from previous page)

```
        # use specified level only
        else
            if checkpid $pid; then
                kill $killlevel $pid >/dev/null 2>&1
                RC=$?
                [ "$RC" -eq 0 ] && success $"$base $killlevel" || failure $"$base
↪$killlevel"
            elif [ -n "${LSB:-}" ]; then
                RC=7 # Program is not running
            fi
        fi
    else
        if [ -n "${LSB:-}" -a -n "$killlevel" ]; then
            RC=7 # Program is not running
        else
            failure $"$base shutdown"
            RC=0
        fi
    fi

    # Remove pid file if any.
    if [ -z "$killlevel" ]; then
        rm -f "${pid_file:-/var/run/$base.pid}"
    fi
    return $RC
}
```

由上述代码可知：关闭进程时，需要再三确定pid文件是否存在，/proc下是否有和pid对应的目录。直到/proc下已经没有了和pid对应的目录时，才表示进程真正杀死了；但此时pid文件仍可能存在，因此还要保证pid文件已被移除

该函数的调用方法如下

```
killproc [-p pidfile] [ -d delay] {program} [-signal]
```

其中

- -p pidfile：用于指定从此文件中获取进程的pid号，不指定时默认从/var/run/$base.pid中获取
- -d delay：指定未使用-signal时的延迟检测时间；有效单位为秒、分、时、日("smhd")，不写时默认为秒
- -signal：用于指定kill发送的信号；如果不指定，则默认先发送TERM信号，在-d delay时间段内仍不断检测是否进程已经被杀死，如果还未死透，则delay超时后发送KILL信号强制杀死

需要明确的是，只有/proc目录下没有了pid对应的目录才算是杀死了； 一般来说，killproc前会判断进程是否已在运行，最后还要删除pid文件和lock文件；当然，killproc函数可以保证pid文件被删除；所以，killproc函数大致会同时配合以下语句用来杀进程

```
status -p ${pidfile} $prog > /dev/null
if [[ $? = 0 ]]; then
        echo -n $"Stopping $prog: "
        killproc -p ${pidfile} -d ${STOP_TIMEOUT} $httpd
else
        echo -n $"Stopping $prog: "
        success
fi
RETVAL=$?
[ $RETVAL -eq 0 ] && rm -f ${lockfile} ${pidfile}
```

同样注意，killproc中已经自带success和failure函数；如果不使用killproc杀进程，则通常会配合action函数或者success、‘‘failure‘‘；大致如下

```
killall $prog ;  usleep 50000 ;  killall $prog
RETVAL=$?
if [ "RETVAL" -ne 0 ];then
    action $"Stopping $prog: " /bin/true
    rm -rf ${lockfile} ${pidfile}
else
    action $"Stoping $prog: " /bin/false
fi
```

以上由于采用的是killall命令，如果采用的是kill命令，则需要先获取进程的pid，在此之前还要检查pid文件是否存在

0x0107 status

status函数用于获取进程的运行状态，有以下几种状态

- ${base} (pid $pid) is running...
- ${base} dead but pid file exists
- ${base} status unknown due to insufficient privileges
- ${base} dead but subsys locked
- ${base} is stopped

status函数定义语句如下(注意：此为CentOS 7上语句，比CentOS 6多了一段systemctl的处理，用于Sysv的status状态向systemd的status状态转换)

```
status() {
    local base pid lock_file= pid_file= binary=

    # Test syntax.
    if [ "$#" = 0 ] ; then
        echo $"Usage: status [-p pidfile] {program}"
        return 1
    fi
    if [ "$1" = "-p" ]; then
        pid_file=$2
        shift 2
    fi
    if [ "$1" = "-l" ]; then
        lock_file=$2
        shift 2
    fi
    if [ "$1" = "-b" ]; then
        if [ -z $pid_file ]; then
            echo $"-b option can be used only with -p"
            echo $"Usage: status -p pidfile -b binary program"
            return 1
        fi
        binary=$2
        shift 2
    fi
    base=${1##*/}

    if [ "$_use_systemctl" = "1" ]; then
        systemctl status ${0##*/}.service
        ret=$?
        # LSB daemons that dies abnormally in systemd looks alive in systemd's␣
↪eyes due to RemainAfterExit=yes
        # lets adjust the reality a little bit
```

(continues on next page)

(continued from previous page)

```
        if systemctl show -p ActiveState ${0##*/}.service | grep -q '=active$' && \
        systemctl show -p SubState ${0##*/}.service | grep -q '=exited$' ; then
            ret=3
        fi
        return $ret
    fi

    # First try "pidof"
    __pids_var_run "$1" "$pid_file" "$binary"
    RC=$?
    if [ -z "$pid_file" -a -z "$pid" ]; then
        pid="$(__pids_pidof "$1")"
    fi
    if [ -n "$pid" ]; then
        echo $"${base} (pid $pid) is running..."
        return 0
    fi

    case "$RC" in
    0)
        echo $"${base} (pid $pid) is running..."
        return 0
        ;;
    1)
        echo $"${base} dead but pid file exists"
        return 1
        ;;
    4)
        echo $"${base} status unknown due to insufficient privileges."
        return 4
        ;;
    esac
    if [ -z "${lock_file}" ]; then
        lock_file=${base}
    fi
    # See if /var/lock/subsys/${lock_file} exists
    if [ -f /var/lock/subsys/${lock_file} ]; then
        echo $"${base} dead but subsys locked"
        return 2
    fi
    echo $"${base} is stopped"
    return 3
}
```

该函数的调用方法如下

```
status [-p pidfile] [-l lockfile] program
# 如果同时提供了-p和-l选项，-l选项必须放在-p选项后面
```

脚本示例

目录

示例脚本

- 写一个脚本，实现如下功能：让用户通过键盘输入一个用户名，如果用户存在，就显示其用户名和UID，否则，就显示用户不存在

```
#!/bin/bash

read -p "please input userName: " userName
if grep "^$userName\>" /etc/passwd &> /dev/null;then
        echo "$userName :`id -u $userName`";
else
        echo "$userName is not exist !!";
fi
```

- 写一脚本，实现如下功能
  - 1、让用户通过键盘输入一个用户名，如果用户不存在就退出
  - 2、如果用户的UID大于等于500，就说明它是普通用户
  - 3、否则，就说明这是管理员或系统用户

```
#!/bin/bash

read -p "please input userName: " userName

if ! grep "^$userName\>" /etc/passwd &> /dev/null;then
        echo "$userName  not exist"
        exit 6
fi

uid=`id -u $userName`
if [ $uid -ge 500 ];then
        echo "The $userName is common user"
else
        echo "The $userName is system user"
fi
```

- 写一脚本，实现如下功能
  - 1、让用户通过键盘输入一个用户名，如果用户不存在就退出
  - 2、如果其UID等于其GID，就说它是个”good guy”
  - 3、否则，就说它是个”bad guy”

```
#!/bin/bash
read -p "please input userName: " userName

if ! grep "^$userName\>" /etc/passwd &> /dev/null;then
        echo "$userName not exist"
        exit 62
fi

if [ `id -u $userName` -eq `id -g $userName` ];then
        echo "$userName is good guy"
else
        echo "$userName is bad guy"
fi
```

- 判断当前系统的所有用户是goodguy还是badguy

```
#!/bin/bash

for userName in `cut -d: -f1 /etc/passwd`;do
        if [ `id -u $userName` -eq `id -g $userName` ];then
                echo "$userName is good guy"
        else
                echo "$userName is bad guy"
```

(continues on next page)

(continued from previous page)

```
        fi
done
```

- 写一个脚本，实现如下功能
  - 1、添加10个用户stu1-stu10；但要先判断用户是否存在
  - 2、如果存在，就用红色显示其已经存大在
  - 3、否则，就添加此用户；并绿色显示
  - 4、最后显示一共添加了几个用户

```
declare -i userCount=0

for i in {1..10};do
        if grep "^stu$i\>" /etc/passwd &> /dev/null;then
                echo -e "\033[31mstu$i\033[0m exist"
        else
                useradd stu$i && echo -e "useradd \033[32mstu$i\033[0m finished"
                let userCount++
        fi
done

echo "Add $userCount users"
```

- 判断当前系统中所有用户是否拥有可登录shell

```
#!/bin/bash

for userName in `cut -d: -f1 /etc/passwd`; do
        if [[ `grep "^$userName\>" /etc/passwd | cut -d: -f7` =~ sh$ ]];then
                echo "login shell user: $userName"
        else
                echo "nologin shell user: $userName"
        fi
done
```

- 写一个脚本，实现如下功能
  - 1.显示如下菜单
    - cpu) show cpu info
    - mem) show memory info
    - quit) quit
  - 2.如果用户选择cpu，则显示/proc/cpuinfo的信息
  - 3.如果用户选择mem，则显示/proc/meminfo的信息
  - 4.如果用户选择quit，则退出，且退出码为5
  - 5.如果用户键入其它字符，则显示未知选项，请重新输入

```
#!/bin/bash

info="cpu) show cpu info\nmem) show memory info\nquit) quit"
while true;do
        echo -e $info

        read -p "Enter your option: " userOption
        userOption=`echo $userOption | tr 'A-Z' 'a-z'`
```

(continues on next page)

(continued from previous page)

```
        if [[ "$userOption" == "cpu" ]];then
                cat /proc/cpuinfo
        elif [[ "$userOption" == "mem" ]];then
                cat /proc/meminfo
        elif [[ "$userOption" == "quit" ]];then
                echo "quit"
                retValue=5
                break
        else
                echo "unkown option"
                retValue=6
        fi
done

[ -z $retValue ] && retValue=0

exit $retValue
```

- 写一个脚本，实现如下功能
  - 1.分别复制/var/log下的文件至/tmp/logs目录中
  - 2.复制目录时，使用cp -r
  - 3.复制文件时，使用cp
  - 4.复制链接文件时，使用cp -d
  - 5.余下的类型，使用cp -a

```
#!/bin/bash

targetDir='/tmp/logs'

[ -e $targetDir ] && mkdir -p $targetDir

for fileName in /var/log/*;do
        if [ -d $fileName ]; then
                copyCmd='cp -r'
        elif [ -f $fileName ]; then
                copyCmd='cp'
        elif [ -h $fileName ]; then
                copyCmd='cp -d'
        else
                copyCmd='cp -a'
        fi

        $copyCmd $fileName $targetDir
done
```

- 写一个脚本，使用形式：`userinfo.sh -u username [-v {1|2}]`
  - -u选项用于指定用户，而后脚本显示用户的UID和GID
  - -v选项后面是1，则显示用户的家目录路径；如果是2，则显示用户的家目录路径和shell

```
#!/bin/bash

[ $# -lt 2 ] && echo "less arguments" && exit 3

if [[ "$1" == "-u" ]]; then
        userName="$2"
        shift 2         # 剔除前2个位置参数
fi
```

(continues on next page)

(continued from previous page)

```
if [[ $# -ge 2 ]] && [ "$1" == "-v" ]; then
        verFlag=$2
fi

verFlag=${verFlag:-0}

if [ -n $verFlag ]; then
        if ! [[ $verFlag =~ [012] ]]; then
                echo "Wrong Parameter"
                echo "Usage: `basename $0` -u UserName -v {1|2}"
                exit 4
        fi
fi

if [ $verFlag -eq 1 ];then
        grep "^$userName" /etc/passwd | cut -d: -f1,3,4,6
elif [ $verFlag -eq 2 ];then
        grep "^$userName" /etc/passwd | cut -d: -f1,3,4,6,7
else
        grep "^$userName" /etc/passwd | cut -d: -f1,3,4
fi
```

- 写一个脚本，实现功能如下
  - 提示用户输入一个用户名，判断用户是否登录了当前系统
  - 如果没有登录，则停止5秒之后，再次判定；直到用户登陆系统，显示用户来了，然后退出

```
#!/bin/bash

read -p "Enter a user name: " userName

# 判断输入是否为空并且是否存在该用户
until [ -n "$userName" ] && id $userName &> /dev/null; do
        read -p "Enter a user name again: " userName
done

until who | grep "^$userName" &> /dev/null; do
        echo "$userName is offline"
        sleep 5
done

echo "$userName is online"
```

- 写一个脚本，实现功能如下
  - 1.提示用户输入一个磁盘设备文件路径不存在或不是一个块设备，则提示用户重新输入，知道输入正确为止，或者输入quit以9为退出码结束脚本
  - 2.提示用户”下面的操作会清空磁盘的数据，并提问是否继续”。如果用户给出字符y或yes，则继续，否则，则提供以8为退出码结束脚本
  - 3.将用户指定的磁盘上的分区清空，而后创建两个分区，大小分别为100M和512M
  - 4.格式化这两个分区
  - 5.将第一个分区挂载至/mnt/boot目录，第二个分区挂载至/mnt/sysroot目录

```
#! /bin/bash
read -p "Enter you dev " devdir
umonut /mnt/boot
umonut /mnt/sysroot
```

(continues on next page)

(continued from previous page)

```
while [[ "$devdir" != "quit" ]];do
        [ -a $devdir ] && [ -b $devdir ]
        if [[ $? -eq 0 ]];then
                read -p "Are you sure[y|yes]: " option
                if [[ "$option" == "y" || "$option" == "yes" ]];then
                        dd if=/dev/zero of=$devdir bs=512 count=1 &> /dev/null
                        echo -e "n\np\n1\n\n+100M\nn\np\n2\n\n+512M\nw" | fdisk
↪$devdir
                        mke2fs -t ext4 ${devdir}1
                        mke2fs -t ext4 ${devdir}2
                        mount ${devdir}1 /mnt/boot
                        mount ${devdir}2 /mnt/sysroot
                        echo "${devdir}1 /mnt/boot ext4 default 0 0" >> /etc/fstab
                        echo "${devdir}2 /mnt/sysroot ext4 default 0 0" >> /etc/
↪fstab
                        exit 7
                else
                        exit 8
                fi
    else
        read -p "Enter you dev again:  "  devdir
    fi
done

exit 9
```

- 写一个脚本，实现功能如下
  - 提示用户输入一个目录路径
  - 显示目录下至少包含一个大写字母的文件名

```
#! /bin/bash

while true; do
        read -p "Enter a directory: " dirname
        [ "$dirname" == "quit" ] && exit 3
        [ -d "$dirname" ] && break || echo "wrong directory..."
done

for filename in $dirname/*;do
        if [[ "$fileName" =~ .*[[:upper:]]{1,}.* ]]; then
        echo "$fileName"
    fi
done
```

- 写一个脚本，实现功能如下(前提是配置好yum源)
  - 1、如果本机没有一个可用的yum源，则提示用户，并退出脚本(4)；如果此脚本非以root用户执行，则显示仅有root才有权限安装程序包，而后退出(3)
  - 2、提示用户输入一个程序包名称，而后使用yum自动安装之；尽可能不输出yum命令执行中的信息；如果安装成功，则绿色显示，否则，红色显示失败
  - 3、如果用户输入的程序包不存在，则显示错误后让用户继续输入
  - 4、如果用户输入quit，则正常退出(0)
  - 5、正常退出前，显示本地共安装的程序包的个数

```
#!/bin/bash
```

(continues on next page)

(continued from previous page)

```
while true;do
        if [ $UID -ne 0 ]; then
                echo "`basename $0` must be running as root"
                exit 3
        fi

        yum repolist &> /dev/null
        if  [[ $? -eq 0 ]];then
                while true; do
                        read -p "Enter a pakage: " pacName
                        if [[ "$pacName" == "quit" ]];then
                                rpm -qa | wc -l
                                exit 0
                fi

                        yum list | grep "^$pacName.*" &> /dev/null
                if [[ $? -eq 0 ]];then
                        yum install $pacName -y &> /dev/null
                        if [[ $? -ne 0 ]];then
                                echo "$pacName install fail"
                        else
                                echo "$pacName install success"
                        fi
                else
                        echo "$pacName is not exist"
                        continue
                fi
                done
        else
                echo "yum repo is not ok!"
                exit 4
        fi
done
```

- 写一个脚本，完成功能如下
  - 1.提示用户输入一个nice值
  - 2.显示指定nice指定进程名及pid
  - 3.提示用户选择要修改nice值的进程的pid和nice值
  - 4.执行修改
  - 5.别退出，继续修改

```
#!/bin/bash

if [[ $UID -eq 0 ]];then
   echo "keyi suibian tiao nice !"
else
   echo "zhineng tiaoda nice !"
fi

while true;do
        read -p "Enter a nice : " nicename
        [ "$nicename" == "quit" ] && exit 3
        /bin/ps axo nice,user,command,pid| grep "^[[:space:]]${nicename}\>"
        read -p "Enter a nice : " niceid
        read -p "Ener a PID :  " pidid
        /usr/bin/renice $niceid $pidid
done
```

- 写一个脚本，实现功能如下：能对/etc/进行打包备份，备份位置为/backup/etc-日期.后缀

- 1.显示如下菜单给用户
  - xz) xz compress
  - gzip) gzip compress
  - bzip2) bzip2 compress
- 2.根据用户指定的压缩工具使用tar打包压缩
- 3.默认为xz，输入错误则需要用户重新输入

```
#!/bin/bash

# 方法一
[ -d /backup ] || mkdir /backup
cat << EOF
xz) xz compress
gzip) gzip compress
bzip2) bzip2 compress
EOF

while true;do
        read -p "Enter a options :" tarname
        [[ "$tarname" == "quit" ]] && exit 5
        tarname=${tarname:-xz}              # tarname为空时给定默认值

        case $tarname in
                xz)
                        tar Jcf /backup/etc-`date +%F-%H-%M-%S`.tar.xz /etc/*
                        break
                        ;;
                gzip)
                        tar zcf /backup/etc-`date +%F-%H-%M-%S`.tar.gz /etc/*
                        break
                        ;;
                bzip2)
                        tar jcf /backup/etc-`date +%F-%H-%M-%S`.tar.bz2 /etc/*
                        break
                        ;;
                *)
                        echo "you Enter is wrong option!"
        esac
done


# 方法二

#!/bin/bash

[ -d /backup ] || mkdir /backup

cat << EOF
plz choose a compress tool:

xz) xz compress
gzip) gzip compress
bzip2) bzip2 compress
EOF

while true; do
        read -p "your optopn: " option
        option=${option:-xz}
```

(continues on next page)

(continued from previous page)

```
        case $option in
        xz)
                compressTool="J"
                suffix='xz'
                break
                ;;
        gzip)
                compressTool="z"
                suffix='gz'
                break
                ;;
        bzip2)
                compressTool="j"
                suffix='bz2'
                break
                ;;
        *)
                echo "wrong option"
                ;;
        esac
done

tar ${compressTool}cf /backup/etc-`date +%F-%H-%M-%S`.tar.$suffix /etc/*
```

实用脚本

shell脚本常用来启动相关系统服务

- *memcached*服务启动脚本

0x00 memcached服务启动脚本

以下是memcached服务启动脚本的示例，是一个非常简单但却非常通用的SysV服务启动脚本

- 关于SysV服务启动脚本的详解请参考：如何写SysV服务管理脚本

```
#!/bin/bash
#
# chkconfig: - 86 14
# description: Distributed memory caching daemon

## Default variables
PORT="11211"
USER="nobody"
MAXCONN="1024"
CACHESIZE="64"
OPTIONS=""

RETVAL=0
prog="/usr/local/memcached/bin/memcached"
desc="Distributed memory caching"
lockfile="/var/lock/subsys/memcached"

. /etc/rc.d/init.d/functions
[ -f /etc/sysconfig/memcached ] && source /etc/sysconfig/memcached

start() {
        echo -n $"Starting $desc (memcached): "
```

(continues on next page)

(continued from previous page)

```
        daemon $prog -d -p $PORT -u $USER -c $MAXCONN -m $CACHESIZE "$OPTIONS"
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && touch $lockfile
        return $RETVAL
}

stop() {
        echo -n $"Shutting down $desc (memcached): "
        killproc $prog
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && rm -f $lockfile
        return $RETVAL
}

restart() {
        stop
        start
}

reload() {
        echo -n $"Reloading $desc ($prog): "
        killproc $prog -HUP
        RETVAL=$?
        echo
        return $RETVAL
}

case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  restart)
        restart
        ;;
  condrestart)
        [ -e $lockfile ] && restart
        RETVAL=$?
        ;;
  reload)
        reload
        ;;
  status)
        status $prog
        RETVAL=$?
        ;;
   *)
        echo $"Usage: $0 {start|stop|restart|reload|condrestart|status}"
        RETVAL=1
esac

exit $RETVAL
```

1.5 通配机制

1.5.1 目录

文件名通配机制

文件名通配机制也称为`globbing`通配机制,它是bash环境的一种特性，用于在shell环境中通配文件名。其通配机制如下：

- `*`：匹配任意长度的任意字符–全文件名部分匹配
- `?`: 匹配单个任意字符–全文件名精确匹配
- `[]`: 匹配指定字符范围内的任意单个字符，不区分大小写
  - `[a-z]`：不区分大小写，可以匹配大写字母
  - `[A-Z]`：不区分大小写，可以匹配小写字母
  - `[0-9]`：匹配0到9任意单个数字
  - `[a-z0-9]`：匹配单个字母或数字
  - `[[:upper:]]`：匹配单个大写字母
  - `[[:lower:]]`：匹配单个小写字母
  - `[[:alpha:]]`：匹配单个大写或小写字母
  - `[[:digit:]]`：匹配单个数字
  - `[[:alnum:]]`：匹配单个字母或数字
  - `[[:space:]]`：匹配单个空格字符
  - `[[:punct:]]`：匹配单个标点符号
- `[^]`: 匹配指定字符范围外的任意单个字符
  - `[^a-z]`：匹配字母之外的单个字符
  - `[^A-Z]`：匹配字母之外的单个字符
  - `[^0-9]`：匹配数字之外的单个字符
  - `[^a-z0-9]`：匹配字母和数字之外的单个字符
  - `[^[:upper:]]`：匹配大写字母之外的单个字符
  - `[^[:lower:]]`：匹配小写字母之外的单个字符
  - `[^[:alpha:]]`：匹配字母之外的单个字符
  - `[^[:digit:]]`：匹配数字之外的单个字符
  - `[^[:alnum:]]`：匹配字母和数字之外的单个字符
  - `[^[:space:]]`：匹配空格字符之外的单个字符
  - `[^[:punct:]]`：匹配标点符号之外的单个字符

正则表达式

参考文档

- 揭开正则表达式的神秘面纱
- 正则表达式手册
- 正则表达式30分钟入门教程

正则表达式，又称为规则表达式(Regular Expression)，通常可以缩写成regex、regexp、regexps、regexes或RE

正则表达式的发展历史

- 美国两位神经生理科学家Warren McCulloch和Walter Pitts研究出了一种用数学方式描述神经网络的方法，该方法将神经系统中的神经元描述成了小而简单的自动控制元
- 1951年，科学家Stephen Kleene在上述研究的基础上发表了《神经网事件的表示法》的论文，该论文引入了正则表达式的概念，将上述的数学控制元称为正则集合的数学符号
- Unix之父Ken Thompson将正则表达式的研究成果应用到计算搜索算法中，将此符号系统引入到编辑器QED以及grep中
- 在之后的几十年里，正则表达式的思想被广泛应用到主流操作系统工具(类Unix、Windows)、主流开发语言(perl、c++、python、JavaScript)等各种应用领域中

正则表达式应用场景：文本处理

- 通过模式进行文本搜索
- 通过模式进行文本替换

接下来将从以下四个方面介绍正则表达式

- 语法：正则表达式的组成成分(字符和元字符)
- 引擎：正则表达式的工作原理(模式匹配)
- 工具：正则表达式的编写和测试工具
- 应用：正则表达式的常见应用

目录

正则表达式语法

参考文档

- 揭开正则表达式的神秘面纱
- 正则表达式30分钟入门教程
- 正则表达式速查表

正则表达式是由普通ASCII字符和正则表达式元字符组合书写出来的一段字符串，这里我们称之为模式

- 普通ASCII字符：大小写字母、数字
- 正则表达式元字符：不表示字符本身的意义，用于额外的功能(字符通配、位置锚定、次数匹配)

要想真正的用好正则表达式，正确理解元字符是非常重要的事情

根据正则表达式元字符本身写法的不同，可将正则表达式分为

- 基本正则表达式(BRE)
- 扩展正则表达式(ERE)
- Perl正则表达式(PCRE)

目录

基本正则表达式

字符通配元字符

- `.`：匹配除`\n`之外的任何单个字符。要匹配包括`\n`在内的任何字符，请使用像`(.|\n)`的模式
- `[]`：匹配指定集合或范围内的任意单个字符
  - `[abc]`：匹配a/b/c中的任意一个字符
  - `[0-9]`、`[[:digit:]]`：匹配0到9任意单个数字
  - `[a-z]`、`[[:lower:]]`：匹配单个小写字母
  - `[A-Z]`、`[[:upper:]]`：匹配单个大写字母
  - `[[:alpha:]]`：匹配单个大写或小写字母
  - `[[:alnum:]]`：匹配单个字母或数字
  - `[[:space:]]`：匹配单个空格字符
  - `[[:punct:]]`：匹配单个标点符号
- `[^]`：匹配指定集合或范围外的任意单个字符
  - `[^a-z]`、`[^[:lower:]]`：匹配小写字母之外的单个字符
  - `[^A-Z]`、`[^[:upper:]]`：匹配大写字母之外的单个字符
  - `[^0-9]`、`[^[:digit:]]`：匹配数字之外的单个字符
  - `[^a-z0-9]`：匹配字母和数字之外的单个字符
  - `[^[:alpha:]]`：匹配字母之外的单个字符
  - `[^[:alnum:]]`：匹配字母和数字之外的单个字符
  - `[^[:space:]]`：匹配空格字符之外的单个字符
  - `[^[:punct:]]`：匹配标点符号之外的单个字符
- `x|y`：匹配x或y
  - `z|food`：匹配z或food
  - `(z|f)ood`：匹配zood或food
- `\cx`：匹配由x指明的控制字符；x的值必须为A-Z或a-z之一。否则，将c视为一个原义的c字符
- `\d`：匹配一个数字字符，等价于`[0-9]`
- `\D`：匹配一个非数字字符，等价于`[^0-9]`
- `\f`：匹配一个换页符，等价于`\x0c`和`\cL`
- `\n`：匹配一个换行符，等价于`\x0a`和`\cJ`
- `\r`：匹配一个回车符，等价于`\x0d`和`\cM`
- `\t`：匹配一个制表符，等价于`\x09`和`\cI`
- `\v`：匹配一个垂直制表符，等价于`\x0b`和`\cK`
- `\s`：匹配任何不可见字符，包括空格、制表符、换页符等等，等价于`[\f\n\r\t\v]`
- `\S`：匹配任何可见字符，等价于`[^\f\n\r\t\v]`
- `\w`：匹配包括下划线的任何单词字符，类似但不等价于`[A-Za-z0-9_]`，这里的单词字符使用Unicode字符集

- \W：匹配任何非单词字符
- \xn：匹配n，其中n为十六进制转义值。十六进制转义值必须为确定的两个数字长
  - \x41：匹配A
  - \x041：等价于\x04&1
- \un：匹配n，其中n是一个用四个十六进制数字表示的Unicode字符
  - \u00A9：匹配版权符号&copy
- \n：标识一个八进制转义值或一个向后引用。如果\n之前至少存在n个获取的子表达式(即至少存在n个分组)，则n为向后引用。否则，如果n为八进制数字0-7，则n为一个八进制转义值
- \nm：标识一个八进制转义值或一个向后引用。如果\nm之前至少存在nm个获取的子表达式(即至少存在nm个分组)，则nm为向后引用。如果\nm之前至少存在n个获取的子表达式，则n为一个后跟文字m的向后引用。如果前面的条件都不满足，若n和m均为八进制数字0-7，则nm将匹配八进制转义值nm。
- \nml：如果n为八进制数字0-7，且m和l均为八进制数字0-7，则匹配八进制转义值nml。

次数通配元字符

作用对象：作用于紧挨着元字符的前面普通单字符、被元字符匹配到的单字符或字符串

- *：匹配前面的子表达式出现任意次
  - 匹配单字符：默认是匹配前一个紧挨次数通配元字符的字符；例如，zo*匹配z以及zoo；等价于{0,}
  - 匹配字符串：使用()将字符串括起来，注意此处需要转义；例如z\(xo\)*
- \+：匹配前面的子表达式出现一次或多次
  - 匹配单字符：默认是匹配前一个紧挨次数通配元字符的字符；例如，zo\+匹配zo以及zoo，但不能匹配z；等价于\{1,\}
  - 匹配字符串：使用()将字符串括起来，注意此处需要转义；例如z\(xo\)\+
- \?：匹配前面的子表达式出现零次或一次
  - 匹配单字符：默认是匹配前一个紧挨次数通配元字符的字符；例如，zo\?匹配zo以及z，但不能匹配zoo
  - 匹配字符串：使用()将字符串括起来，注意此处需要转义；例如do\(es\)\?匹配does或does中的do；等价于\{0,1\}
- \{n\}：n是一个非负整数，匹配前面的子表达式出现n次
  - 匹配单字符：默认是匹配前一个紧挨次数通配元字符的字符；例如，o\{2\}不能匹配Bob中的o，但是能匹配food中的两个o
  - 匹配字符串：使用()将字符串括起来，注意此处需要转义；例如do\(es\)\{2\}
- \{n,\}：n是一个非负整数，匹配前面的子表达式至少出现n次
  - 匹配单字符：默认是匹配前一个紧挨次数通配元字符的字符；例如，o\{2,\}不能匹配Bob中的o，但能匹配fooooood中的所有o。o\{1,\}等价于o+。o\{0,\}则等价于o*
  - 匹配字符串：使用()将字符串括起来，注意此处需要转义；例如do\(es\)\{2,\}
- \{n,m\}：m和n均为非负整数，其中n<=m。匹配前面的子表达式至少出现n次至多出现m次
  - 匹配单字符：默认是匹配前一个紧挨次数通配元字符的字符；例如，o\{1,3\}匹配fooooood中的前三个o。o\{0,1\}等价于o\?请注意逗号和两个数之间不能有空格
  - 匹配字符串：使用()将字符串括起来，注意此处需要转义；例如do\(es\)\{1,3\}
- 注意：

  - 当?紧跟在任何一个其他限制符*、\+、\?、\{n\}、\{n,\}、\{n,m\}后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串oooo，o\+?将匹配单个o，而o\+将匹配所有o

位置锚定元字符

作用对象：作用于紧挨着元字符的前面或后面普通单字符、被元字符匹配到的单字符或字符串

- ^：锚定输入字符串的开始位置(即锚定行首)。如果设置了RegExp对象的Multiline多行匹配属性，^也锚定\n或\r之后的位置
- $：锚定输入字符串的结束位置(即锚定行尾)。如果设置了RegExp对象的Multiline多行匹配属性，$也锚定\n或\r之前的位置
  - ^$表示空白行
- \b：锚定一个单词边界(首部或尾部)，也就是指单词和空格间的位置
  - er\b可以匹配never中的er，但不能匹配verb中的er
  - \ber\b只能匹配er
- \B：锚定非单词边界，即不是首部也不是尾部
  - er\B能匹配verb中的er，但不能匹配never中的er
- \<：锚定单词(word)首部
- \>：锚定单词(word)尾部
  - \<the\>能够匹配字符串for the wise中的the，但是不能匹配字符串otherwise中的the

分组引用元字符

作用对象：被pattern匹配到的字符集合(pattern是由字符通配元字符、次数通配元字符、位置锚定元字符组合而成)

- \(pattern\)：匹配pattern并获取这一匹配。并将匹配到的字符保存到一个临时区域Matches集合或VBScript中的SubMatches集合(一个正则表达式中最多可以保存9个)
  - 在shell环境中可以使用\1到 \9基于分组字符串的位置对分组字符串整体依次进行引用(依次引用前面被自左往右的第1到第9个左括号以及与之对应的右括号中的模式匹配到的内容)
  - 在JavaScript环境中可以使用$0到$9基于分组字符串的位置对分组字符串整体依次进行引用(依次引用前面被自左往右的第1到第9个左括号以及与之对应的右括号中的模式匹配到的内容)
- \(?:pattern\)：匹配pattern但不获取匹配结果，也就是说这是一个非获取匹配，不对匹配到的内容做临时存储供以后使用。这在使用或字符(|)来组合一个模式时很有用
  - industr\(?:y|ies\)就是一个比industry|industries更简略的表达式
- \(?=pattern\)：正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，不对匹配到的内容做临时存储供以后使用
  - Windows\(?=95|98|NT|2000\)能匹配Windows2000中的Windows，但不能匹配Windows3.1中的Windows。预查不消耗字符，也就是说，在一个匹配发生后，立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始
- \(?!pattern\)：正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，不对匹配到的内容做临时存储供以后使用
  - Windows\(?!95|98|NT|2000\)能匹配Windows3.1中的Windows，但不能匹配Windows2000中的Windows。预查不消耗字符，也就是说，在一个匹配发生后，立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始

- \(?<=pattern\)：反向肯定预查，与正向肯定预查类拟，只是方向相反
  - \(?<=95|98|NT|2000\)Windows能匹配2000Windows中的Windows，但不能匹配3.1Windows中的Windows
- \(?<!pattern\)：反向否定预查，与正向否定预查类拟，只是方向相反
  - \(?<!95|98|NT|2000\)Windows能匹配3.1Windows中的Windows，但不能匹配2000Windows中的Windows

温馨提示

- 所有没有被\转义的元字符如果想表示字符本身含义，则需要添加\转义符进行转义
- 所有已经被\转义的元字符如果想表示字符本身含义，则需要去掉\转义符取消转义

扩展正则表达式

字符通配元字符

- .：匹配除\n之外的任何单个字符。要匹配包括\n在内的任何字符，请使用像(.|\n)的模式
- []：匹配指定集合或范围内的任意单个字符
  - [abc]：匹配a/b/c中的任意一个字符
  - [0-9]、[[:digit:]]：匹配0到9任意单个数字
  - [a-z]、[[:lower:]]：匹配单个小写字母
  - [A-Z]、[[:upper:]]：匹配单个大写字母
  - [[:alpha:]]：匹配单个大写或小写字母
  - [[:alnum:]]：匹配单个字母或数字
  - [[:space:]]：匹配单个空格字符
  - [[:punct:]]：匹配单个标点符号
- [^]：匹配指定集合或范围外的任意单个字符
  - [^a-z]、[^[:lower:]]：匹配小写字母之外的单个字符
  - [^A-Z]、[^[:upper:]]：匹配大写字母之外的单个字符
  - [^0-9]、[^[:digit:]]：匹配数字之外的单个字符
  - [^a-z0-9]：匹配字母和数字之外的单个字符
  - [^[:alpha:]]：匹配字母之外的单个字符
  - [^[:alnum:]]：匹配字母和数字之外的单个字符
  - [^[:space:]]：匹配空格字符之外的单个字符
  - [^[:punct:]]：匹配标点符号之外的单个字符
- x|y：匹配x或y
  - z|food：匹配z或food
  - (z|f)ood：匹配zood或food
- \cx：匹配由x指明的控制字符；x的值必须为A-Z或a-z之一。否则，将c视为一个原义的c字符
- \d：匹配一个数字字符，等价于[0-9]
- \D：匹配一个非数字字符，等价于[^0-9]

- \f：匹配一个换页符，等价于\x0c和\cL
- \n：匹配一个换行符，等价于\x0a和\cJ
- \r：匹配一个回车符，等价于\x0d和\cM
- \t：匹配一个制表符，等价于\x09和\cI
- \v：匹配一个垂直制表符，等价于\x0b和\cK
- \s：匹配任何不可见字符，包括空格、制表符、换页符等等，等价于[\f\n\r\t\v]
- \S：匹配任何可见字符，等价于[^\f\n\r\t\v]
- \w：匹配包括下划线的任何单词字符，类似但不等价于[A-Za-z0-9_]，这里的单词字符使用Unicode字符集
- \W：匹配任何非单词字符
- \xn：匹配n，其中n为十六进制转义值。十六进制转义值必须为确定的两个数字长
  - \x41：匹配A
  - \x041：等价于\x04&1
- \un：匹配n，其中n是一个用四个十六进制数字表示的Unicode字符
  - \u00A9：匹配版权符号&copy
- \n：标识一个八进制转义值或一个向后引用。如果\n之前至少存在n个获取的子表达式(即至少存在n个分组)，则n为向后引用。否则，如果n为八进制数字0-7，则n为一个八进制转义值
- \nm：标识一个八进制转义值或一个向后引用。如果\nm之前至少存在nm个获取的子表达式(即至少存在nm个分组)，则nm为向后引用。如果\nm之前至少存在n个获取的子表达式，则n为一个后跟文字m的向后引用。如果前面的条件都不满足，若n和m均为八进制数字0-7，则nm将匹配八进制转义值nm。
- \nml：如果n为八进制数字0-7，且m和l均为八进制数字0-7，则匹配八进制转义值nml。

次数通配元字符

作用对象：作用于紧挨着元字符的前面普通单字符、被元字符匹配到的单字符或字符串

- *：匹配前面的子表达式出现任意次
  - zo*匹配z以及zoo；等价于{0,}
- +：匹配前面的子表达式出现一次或多次
  - zo+匹配zo以及zoo，但不能匹配z；等价于{1,}
- ?：匹配前面的子表达式出现零次或一次
  - do(es)?匹配does或does中的do；等价于{0,1}
- {n}：n是一个非负整数，匹配前面的子表达式出现n次
  - o{2}不能匹配Bob中的o，但是能匹配food中的两个o
- {n,}：n是一个非负整数，匹配前面的子表达式至少出现n次
  - o{2,}不能匹配Bob中的o，但能匹配fooooood中的所有o。o{1,}等价于o+。o{0,}则等价于o*
- {n,m}：m和n均为非负整数，其中n<=m。匹配前面的子表达式至少出现n次至多出现m次
  - o{1,3}匹配fooooood中的前三个o。o{0,1}等价于o?请注意逗号和两个数之间不能有空格
- 注意：

  - 当?紧跟在任何一个其他限制符*、+、?、{n}、{n,}、{n,m}后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串oooo，o+?将匹配单个o，而o+将匹配所有o

位置锚定元字符

作用对象：作用于紧挨着元字符的前面或后面普通单字符、被元字符匹配到的单字符或字符串

- ^：锚定输入字符串的开始位置(即锚定行首)。如果设置了RegExp对象的Multiline多行匹配属性，^也锚定\n或\r之后的位置
- $：锚定输入字符串的结束位置(即锚定行尾)。如果设置了RegExp对象的Multiline多行匹配属性，$也锚定\n或\r之前的位置
  - ^$表示空白行
- \b：锚定一个单词边界(首部或尾部)，也就是指单词和空格间的位置
  - er\b可以匹配never中的er，但不能匹配verb中的er
  - \ber\b只能匹配er
- \B：锚定非单词边界，即不是首部也不是尾部
  - er\B能匹配verb中的er，但不能匹配never中的er
- \<：锚定单词(word)首部
- \>：锚定单词(word)尾部
  - \<the\>能够匹配字符串for the wise中的the，但是不能匹配字符串otherwise中的the

分组引用元字符

作用对象：被pattern匹配到的字符集合(pattern是由字符通配元字符、次数通配元字符、位置锚定元字符组合而成)

- (pattern)：匹配pattern并获取这一匹配。并将匹配到的字符保存到一个临时区域Matches集合或VBScript中的SubMatches集合(一个正则表达式中最多可以保存9个)
  - 在shell环境中可以使用\1到 \9基于分组字符串的位置对分组字符串整体依次进行引用(依次引用前面被自左往右的第1到第9个左括号以及与之对应的右括号中的模式匹配到的内容)
  - 在JavaScript环境中可以使用$0到$9基于分组字符串的位置对分组字符串整体依次进行引用(依次引用前面被自左往右的第1到第9个左括号以及与之对应的右括号中的模式匹配到的内容)
- (?:pattern)：匹配pattern但不获取匹配结果，也就是说这是一个非获取匹配，不对匹配到的内容做临时存储供以后使用。这在使用或字符(|)来组合一个模式时很有用
  - industr(?:y|ies)就是一个比industry|industries更简略的表达式
- (?=pattern)：正向肯定预查，在任何匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，不对匹配到的内容做临时存储供以后使用
  - Windows(?=95|98|NT|2000)能匹配Windows2000中的Windows，但不能匹配Windows3.1中的Windows。预查不消耗字符，也就是说，在一个匹配发生后，立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始
- (?!pattern)：正向否定预查，在任何不匹配pattern的字符串开始处匹配查找字符串。这是一个非获取匹配，不对匹配到的内容做临时存储供以后使用
  - Windows(?!95|98|NT|2000)能匹配Windows3.1中的Windows，但不能匹配Windows2000中的Windows。预查不消耗字符，也就是说，在一个匹配发生后，立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始

- (?<=pattern)：反向肯定预查，与正向肯定预查类拟，只是方向相反
  - (?<=95|98|NT|2000)Windows能匹配2000Windows中的Windows，但不能匹配3.1Windows中的Windows
- (?<!pattern)：反向否定预查，与正向否定预查类拟，只是方向相反
  - (?<!95|98|NT|2000)Windows能匹配3.1Windows中的Windows，但不能匹配2000Windows中的Windows

正则表达式引擎

目录

shell环境引擎

python环境引擎

perl环境引擎

c++环境引擎

参考文档

- cpp正则表达式引擎

正则表达式工具

参考文档

- 正则表达式工具

正则表达式应用

目录

shell环境应用

```
1、显示/proc/meminfo文件中以大小写s开头的行；
# grep "^[sS]" /proc/meminfo
# grep -i "^s" /proc/meminfo
2、取出默认shell为非bash的用户；
# grep -v "bash$" /etc/passwd | cut -d: -f1
3、取出默认shell为bash的且其ID号最大的用户；
# grep "bash$" /etc/passwd | sort -n -t: -k3 | tail -1 | cut -d: -f1
4、显示/etc/rc.d/rc.sysinit文件中，以#开头，后面跟至少一个空白字符，而后又有至少一个非空白字符的行；
# grep "^#[[:space:]]\{1,\}[^[:space:]]\{1,\}" /etc/rc.d/rc.sysinit
5、显示/boot/grub/grub.conf中以至少一个空白字符开头的行；
# grep "^[[:space:]]\{1,\}[^[:space:]]\{1,\}" /boot/grub/grub.conf
6、找出/etc/passwd文件中一位数或两位数；
# grep --color=auto "\<[0-9]\{1,2\}\>" /etc/passwd
7、找出ifconfig命令结果中的1到255之间的整数；
8、查看当前系统上root用户的所有信息；
# grep "^root\>" /etc/passwd
```

1.6 数据库

参考文档

- 数据库系列大纲

1.7 网站架构

参考文档

- 网站架构系列大纲