实验四 Shell基础与脚本编程

实验4.1 Shell基础

1. 实验目的

了解Linux Shell（Bash）的基本功能，通过完成本实验，掌握下列技能：

1. 使用通配符扩展文件名
2. 重定向标准输入，标准输出和标准错误
3. 使用管道将一个进程的输出作为另一进程程的输入
4. 执行命令分组和行继续

2. 实验环境

2.1 软件环境：

1. CentOS Linux release 7.9.2009
2. VMware Workstation Pro 15
3. MobaXterm v20.0

2.2 硬件环境：

1. 阿里云云服务器ECS，共享计算型实例，配置（CPU 1核 \ 内存 2GB \ 带宽 1Mbps）
2. 个人笔记本，配置（建议内存>4G)

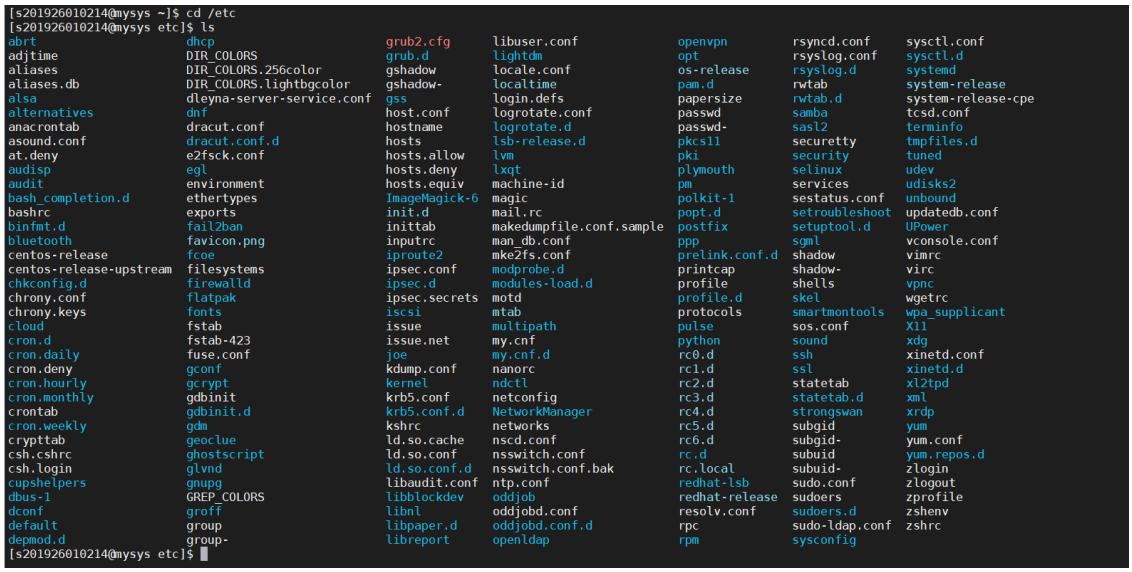
3. 实验步骤和方法

3.1 通配符

1. 打开终端软件ssh会话，连接云服务器ECS，使用账号和密码登录。

> 备注：如果在本地虚拟机的图形环境中实验，请以tux1身份登录，并打开终端仿真器

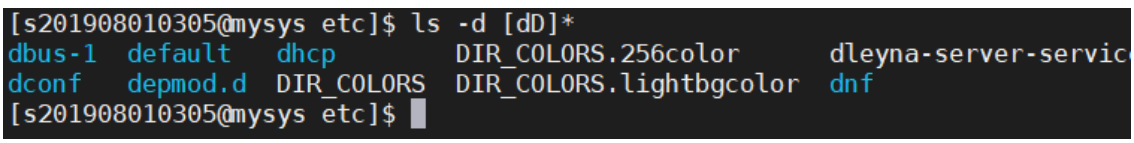
2. 转到/etc目录，并在此处列出所有文件。



3. 使用带通配符的ls列出文件名：

a. 以conf结尾

b. 以d或D开头



c. 在第五个位置包含一个o

```
[s201926010214@mysys etc]$ ls -d ????o*
centos-release           depmod.d     ld.so.conf.d    netconfig       oddjob          plymouth   rc.local        shadow-
centos-release-upstream  ld.so.cache  logrotate.conf  NetworkManager  oddjobd.conf    protocols  setroubleshoot  sysconfig
chkconfig.d              ld.so.conf   logrotate.d     networks        oddjobd.conf.d  python     shadow          zlogout
[s201926010214@mysys etc]$
```

d. 包含单词制表符（大写和小写字符的任意组合）

```
[s201926010214@mysys etc]$ ls -d *[tT][aA][bB]*
anacrontab  crypttab  fstab-423  mtab   rwtab.d   statetab.d
crontab     fstab     inittab    rwtab  statetab
[s201926010214@mysys etc]$
```

e. 以数字结尾

```
[s201926010214@mysys etc]$ ls -d *[0-9]
dbus-1  fstab-423  ImageMagick-6  iproute2  pkcs11  polkit-1  sasl2  udisks2  X11
```

f. 不以数字结尾

```
[s201926010214@mysys etc]$ ls -d *[!0-9]
abrt                        init.d                  rc2.d
adjtime                     inittab                 rc3.d
aliases                     inputrc                 rc4.d
aliases.db                  ipsec.conf              rc5.d
alsa                        ipsec.d                 rc6.d
alternatives                ipsec.secrets           rc.d
anacrontab                  iscsi                   rc.local
asound.conf                 issue                   redhat-lsb
at.deny                     issue.net               redhat-release
audisp                      joe                     resolv.conf
audit                       kdump.conf              rpc
bash_completion.d           kernel                  rpm
bashrc                      krb5.conf               rsyncd.conf
binfmt.d                    krb5.conf.d             rsyslog.conf
bluetooth                   kshrc                   rsyslog.d
centos-release              ld.so.cache             rwtab
centos-release-upstream     ld.so.conf              rwtab.d
chkconfig.d                 ld.so.conf.d            samba
chrony.conf                 libaudit.conf           securetty
chrony.keys                 libblockdev             security
cloud                       libnl                   selinux
cron.d                      libpaper.d              services
cron.daily                  libreport               sestatus.conf
cron.deny                   libuser.conf            setroubleshoot
cron.hourly                 lightdm                 setuptool.d
cron.monthly                locale.conf             sgml
crontab                     localtime               shadow
cron.weekly                 login.defs              shadow-
crypttab                    logrotate.conf          shells
csh.cshrc                   logrotate.d             skel
csh.login                   lsb-release.d           smartmontools
cupshelpers                 lvm                     sos.conf
dconf                       lxqt                    sound
default                     machine-id              ssh
depmod.d                    magic                   ssl
dhcp                        mail.rc                 statetab
DIR_COLORS                  makedumpfile.conf.sample statetab.d
```

> 笔记：

> 请注意，通配符扩展由shell程序完成。如果匹配的文件名之一是目录名，则默认情况下ls会列出该目录的内容，而不是文件名本身。要避免这种情况，请使用-d选项。

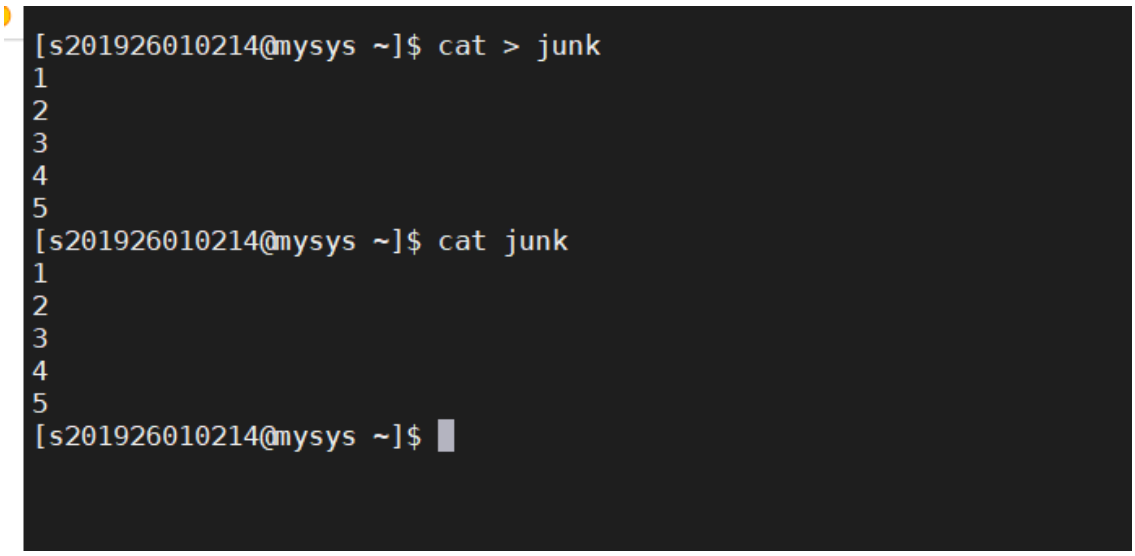
4. 如果执行命令ls -d？[！ y] * [e-g]，会发生什么？可以匹配的最短文件名是什么？执行此命令以验证答案。

```
[s201926010214@mysys etc]$ ls -d ?[!y]*[e-g]
adjtime                    groff          logrotate.conf            rsyncd.conf
asound.conf                grub2.cfg      makedumpfile.conf.sample  rsyslog.conf
centos-release             host.conf      man_db.conf               sestatus.conf
chrony.conf                hostname       mke2fs.conf               sos.conf
dconf                      ipsec.conf     netconfig                 sudo.conf
dleyna-server-service.conf issue          nscd.conf                 sudo-ldap.conf
dnf                        joe            nsswitch.conf             tcsd.conf
dracut.conf                kdump.conf     ntp.conf                  updatedb.conf
e2fsck.conf                krb5.conf      oddjobd.conf              vconsole.conf
favicon.png                ld.so.cache    os-release                xdg
fcoe                       ld.so.conf     papersize                 xinetd.conf
fuse.conf                  libaudit.conf  profile                   yum.conf
gconf                      libuser.conf   pulse                     zprofile
geoclue                    locale.conf    redhat-release
gnupg                      localtime      resolv.conf
[s201926010214@mysys etc]$
```

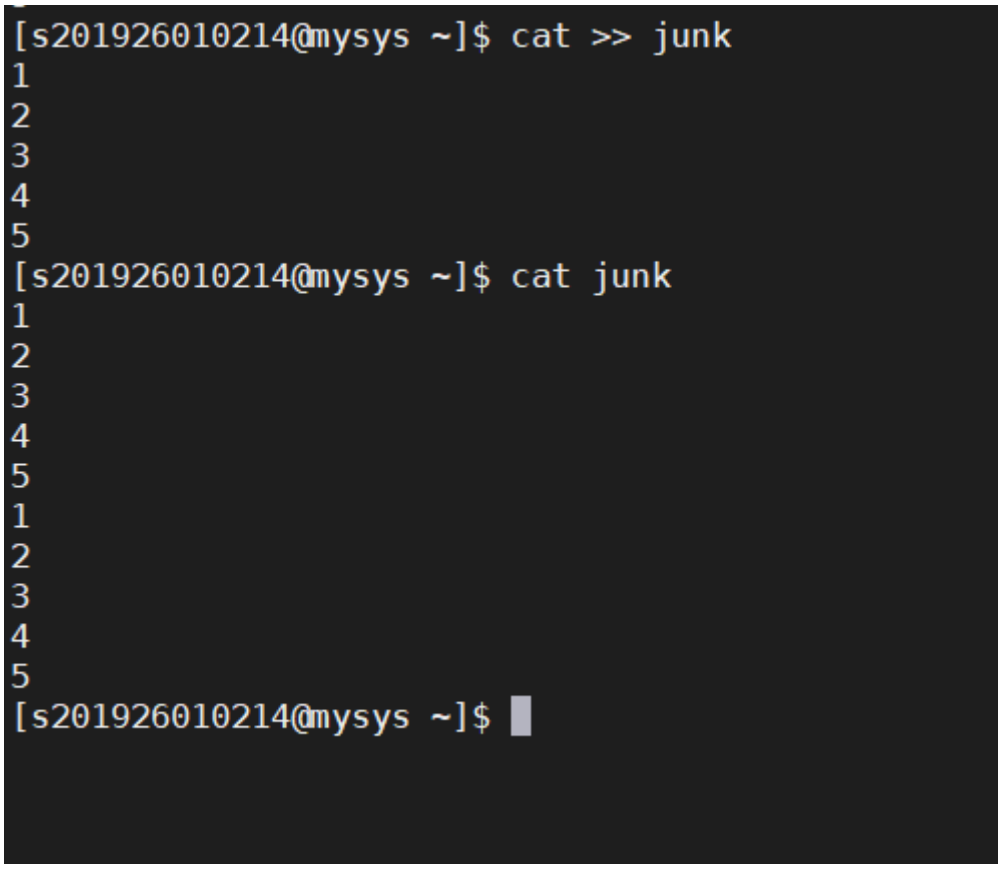
5. 返回主目录。

3.2 重定向

1. 使用cat命令和重定向符来创建一个名为junk的文件，其中包含几行文本。输入几行后，结束对cat 命令的输入并返回到shell提示符。然后查看刚刚创建的文件的内容。



2. 使用重定向符将更多行添加到junk文件。然后查看junk文件的内容，并检查保存在此文件中的所有行是否都存在。



3.3 管道，tee和过滤器

1. 计算当前目录中的文件数。使用wc命令，不要手动计算文件数。

```
[s201926010214@mysys ~]$ ls | wc -l
2
```

2. 执行如下命令计算文件数并保存结果, 查看temfile文件内容，与上一条命令有何差异?

```
[s201926010214@mysys ~]$ ls > tempfile
[s201926010214@mysys ~]$ wc -l tempfile
3 tempfile
[s201926010214@mysys ~]$ rm tempfile
[s201926010214@mysys ~]$
```

3. 在对文件进行计数之前，请使用ls命令并将输出保存到名为tempfile2的文件中。

```
[s201926010214@mysys ~]$ ls | tee tempfile2 | wc -l
3
[s201926010214@mysys ~]$
```

4. 使用sed命令更改ls -l /etc/命令的输出，以便看起来您拥有/etc中的所有文件。使用和不使用全局选项都可以执行此操作。有什么区别?

```
[s201926010214@mysys ~]$ ls -l /etc | sed s/root/tux1/
total 1736
drwxr-xr-x   3 tux1 root     4096 Mar 18 21:51 abrt
-rw-r--r--.  1 tux1 root       18 Dec 25  2019 adjtime
-rw-r--r--   1 tux1 root     1529 Apr  1  2020 aliases
-rw-r--r--   1 tux1 root    12288 May  7  2020 aliases.db
drwxr-xr-x   2 tux1 root     4096 May  2  2020 alsa
drwxr-xr-x.  2 tux1 root     4096 Mar 28 08:31 alternatives
-rw-------.  1 tux1 root      541 Aug  9  2019 anacrontab
-rw-r--r--.  1 tux1 root       55 Aug  8  2019 asound.conf
-rw-r--r--   1 tux1 root        1 Oct 31  2018 at.deny
drwxr-x---.  3 tux1 root     4096 Dec 25  2019 audisp
drwxr-x---.  3 tux1 root     4096 Dec 25  2019 audit
drwxr-xr-x.  2 tux1 root     4096 Mar 18 21:51 bash_completion.d
-rw-r--r--   1 tux1 root     2853 Apr  1  2020 bashrc
drwxr-xr-x.  2 tux1 root     4096 Feb  3 00:34 binfmt.d
drwxr-xr-x   2 tux1 root     4096 Mar 18 21:51 bluetooth
-rw-r--r--   1 tux1 root       37 Nov 23 23:08 centos-release
-rw-r--r--   1 tux1 root       51 Nov 23 23:08 centos-release-upstream
drwxr-xr-x.  2 tux1 root     4096 Oct 13  2020 chkconfig.d
-rw-r--r--.  1 tux1 root     1937 Feb 27  2020 chrony.conf
-rw-r-----.  1 tux1 chrony    481 Aug  8  2019 chrony.keys
drwxr-xr-x   4 tux1 root     4096 Dec 25  2019 cloud
drwxr-xr-x.  2 tux1 root     4096 Mar 18 21:51 cron.d
drwxr-xr-x.  2 tux1 root     4096 May  7  2020 cron.daily
-rw-------.  1 tux1 root        0 Aug  9  2019 cron.deny
drwxr-xr-x.  2 tux1 root     4096 Jun 10  2014 cron.hourly
```

```
[s201926010214@mysys ~]$ ls -l /etc | sed s/root/tux1/g
total 1736
drwxr-xr-x   3 tux1 tux1     4096 Mar 18 21:51 abrt
-rw-r--r--.  1 tux1 tux1       18 Dec 25  2019 adjtime
-rw-r--r--   1 tux1 tux1     1529 Apr  1  2020 aliases
-rw-r--r--   1 tux1 tux1    12288 May  7  2020 aliases.db
drwxr-xr-x   2 tux1 tux1     4096 May  2  2020 alsa
drwxr-xr-x.  2 tux1 tux1     4096 Mar 28 08:31 alternatives
-rw-------.  1 tux1 tux1      541 Aug  9  2019 anacrontab
-rw-r--r--.  1 tux1 tux1       55 Aug  8  2019 asound.conf
-rw-r--r--   1 tux1 tux1        1 Oct 31  2018 at.deny
drwxr-x---.  3 tux1 tux1     4096 Dec 25  2019 audisp
drwxr-x---.  3 tux1 tux1     4096 Dec 25  2019 audit
drwxr-xr-x.  2 tux1 tux1     4096 Mar 18 21:51 bash_completion.d
-rw-r--r--   1 tux1 tux1     2853 Apr  1  2020 bashrc
drwxr-xr-x.  2 tux1 tux1     4096 Feb  3 00:34 binfmt.d
drwxr-xr-x   2 tux1 tux1     4096 Mar 18 21:51 bluetooth
-rw-r--r--   1 tux1 tux1       37 Nov 23 23:08 centos-release
-rw-r--r--   1 tux1 tux1       51 Nov 23 23:08 centos-release-upstream
drwxr-xr-x.  2 tux1 tux1     4096 Oct 13  2020 chkconfig.d
-rw-r--r--.  1 tux1 tux1     1937 Feb 27  2020 chrony.conf
-rw-r-----.  1 tux1 chrony    481 Aug  8  2019 chrony.keys
drwxr-xr-x   4 tux1 tux1     4096 Dec 25  2019 cloud
drwxr-xr-x.  2 tux1 tux1     4096 Mar 18 21:51 cron.d
drwxr-xr-x.  2 tux1 tux1     4096 May  7  2020 cron.daily
-rw-------.  1 tux1 tux1        0 Aug  9  2019 cron.deny
drwxr-xr-x.  2 tux1 tux1     4096 Jun 10  2014 cron.hourly
drwxr-xr-x.  2 tux1 tux1     4096 Jun 10  2014 cron.monthly
```

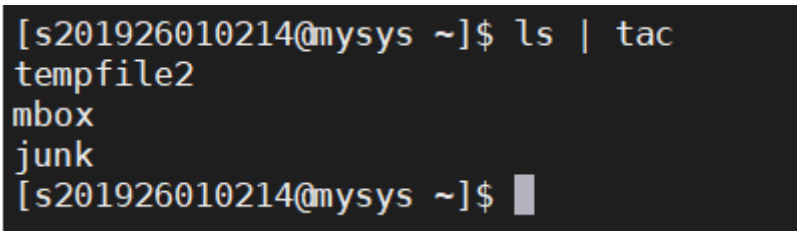
5. 使用awk命令显示/etc目录中所有文件的权限和名称。

```
[s201926010214@mysys ~]$ ls -l /etc | awk '{print $1 "\t" $9}'
total
drwxr-xr-x      abrt
-rw-r--r--.     adjtime
-rw-r--r--      aliases
-rw-r--r--      aliases.db
drwxr-xr-x      alsa
drwxr-xr-x.     alternatives
-rw-------.     anacrontab
-rw-r--r--.     asound.conf
-rw-r--r--      at.deny
drwxr-x---.     audisp
drwxr-x---.     audit
drwxr-xr-x.     bash_completion.d
-rw-r--r--      bashrc
drwxr-xr-x.     binfmt.d
drwxr-xr-x      bluetooth
-rw-r--r--      centos-release
-rw-r--r--      centos-release-upstream
drwxr-xr-x.     chkconfig.d
-rw-r--r--.     chrony.conf
-rw-r-----.     chrony.keys
drwxr-xr-x      cloud
drwxr-xr-x.     cron.d
drwxr-xr-x.     cron.daily
-rw-------.     cron.deny
drwxr-xr-x.     cron.hourly
drwxr-xr-x.     cron.monthly
-rw-r--r--.     crontab
drwxr-xr-x.     cron.weekly
```

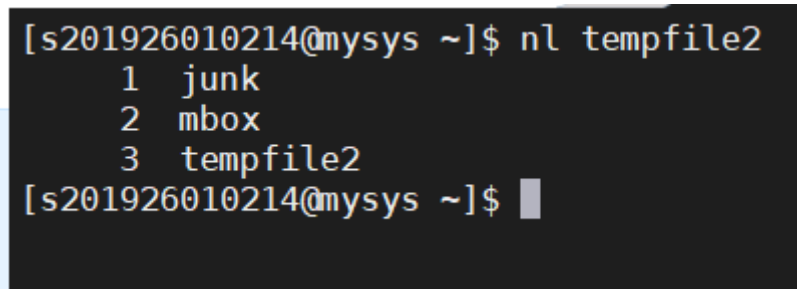
> 笔记：

> RHEL和SUSE之间的区别在于ls命令输出的默认日期格式。在RHEL中，日期写为“ 11月12日”，计 为两列；在SUSE中，日期写为“ 2011-11-12”，计为一列。您可以使用--time-style = something选 项更改ls的这种行为，其中某些东西例如是long-iso。如果将来要编写必须具有可移植性的Shell脚本，请记住这些注意事项！

6. 使用tac命令以相反的顺序显示ls命令的输出。



7. 使用nl命令对tempfile2的行进行编号。



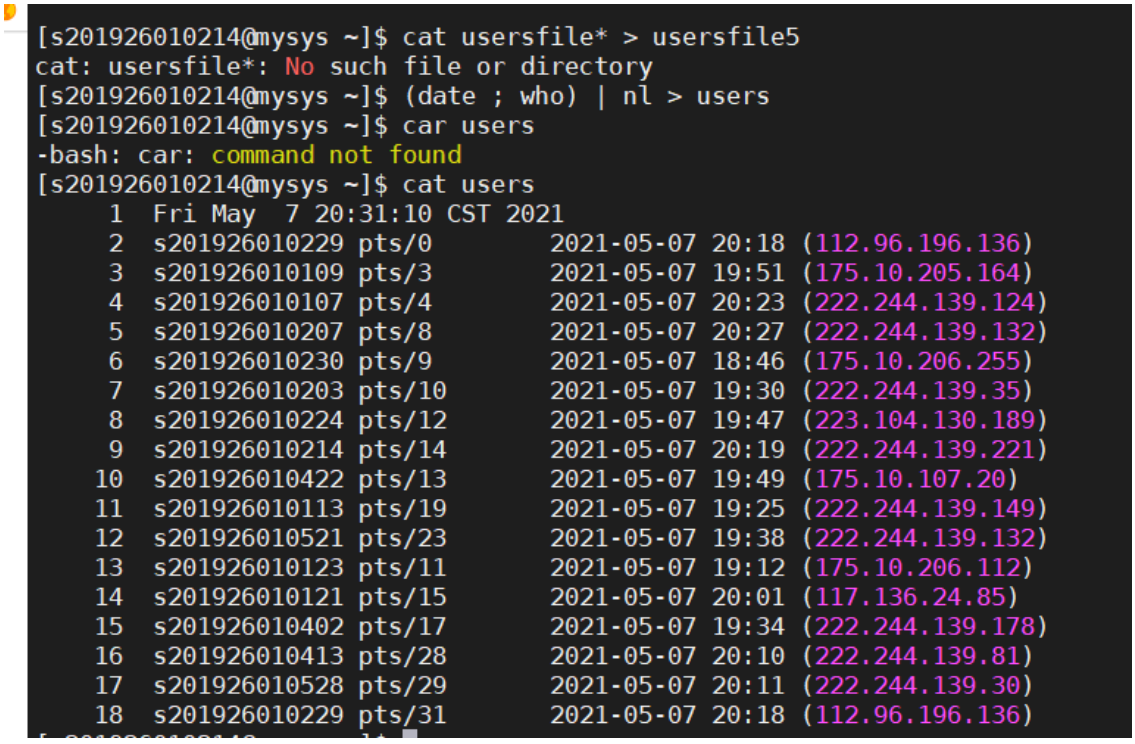
8. 使用pr命令为打印机格式化tempfile2。

9. 将“文件和目录权限”练习中的所有usersfile文件合并为一个大文件，名为usersfile5。检查此文件是 否与原始的users文件相同。

```
$ cat usersfile* > usersfile5
$ diff usersfile usersfile5
```

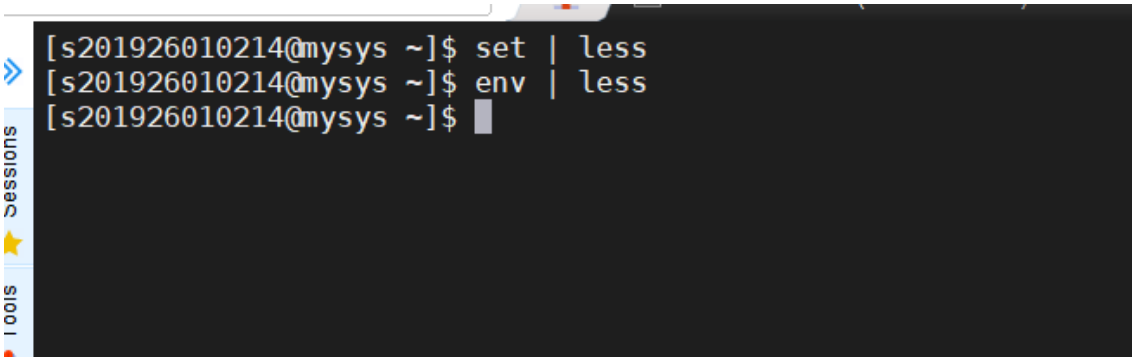
3.4 命令分组

1. 作为一项命令，显示当前系统日期和所有登录的用户，并在对行进行编号后将所有这些保存到一个文件中。检查您的输出。

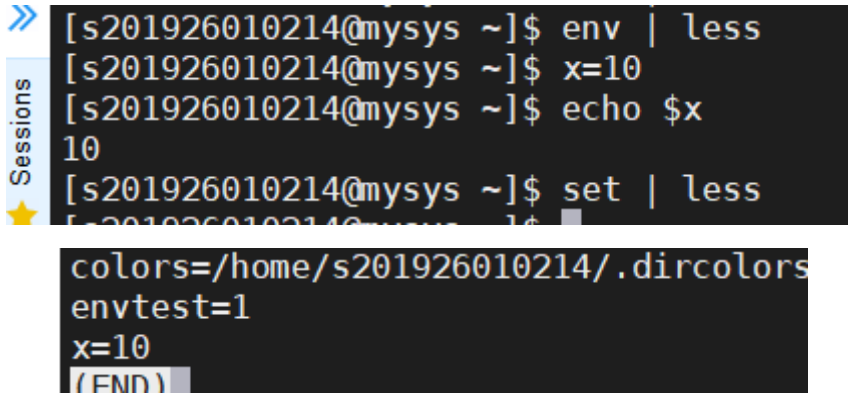


3.5 进程环境

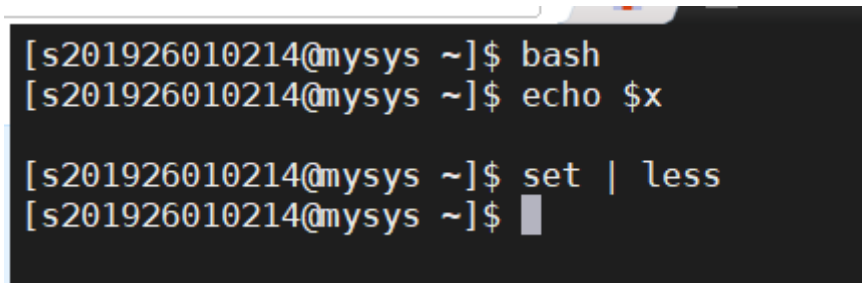
1. 显示当前流程环境中定义的所有变量，同时显示当前导出的所有变量。

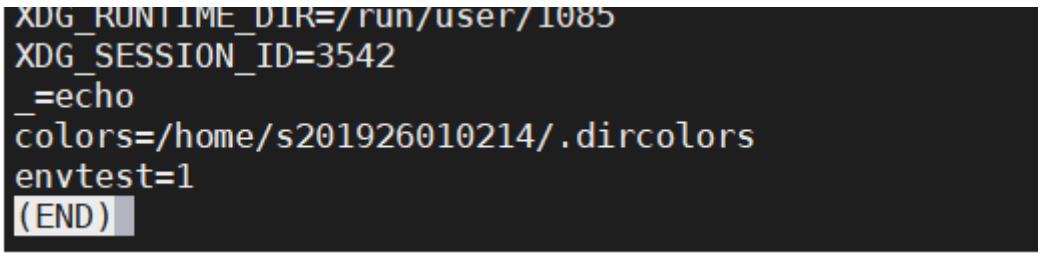


2. 创建变量x并将其值设置为10。检查变量的值。同样，显示所有当前变量和导出的变量。



3. 启动一个子shell 。检查以查看变量x在子shell 中包含什么值。x的值是多少？列出子shell 程序的当前变量。





4. 将x的值设置为500，然后返回到父Shell。 x的当前值是多少?

```
[s201926010214@mysys ~]$ x=500
[s201926010214@mysys ~]$ exit
exit
[s201926010214@mysys ~]$ echo $x
10
[s201926010214@mysys ~]$
```

5. 确保子Shell继承变量x。 通过创建一个子shell并检查变量x的值来验证这一点。之后，退出子shell。

```
[s201926010214@mysys ~]$ export x
[s201926010214@mysys ~]$ env | less
[s201926010214@mysys ~]$ bash
[s201926010214@mysys ~]$ echo $x
10
[s201926010214@mysys ~]$ exit
exit
[s201926010214@mysys ~]$
```

```
XDG_DATA_DIRS=/home/s201926010214/.local/share/flatpak/expo
orts/share:/usr/local/share:/usr/share
SSH_CONNECTION=222.244.139.221 45392 172.19.143.125 22
LESSOPEN=||/usr/bin/lesspipe.sh %s
x=10
:
```

实验4.2 Shell脚本编程

1. 实验目的

在使用Linux一段时间后，学生会发现要自定义的环境的某些特征以及要自动执行的一些定期执行的任 务。

本实验向学生介绍一些更常用的结构，这些结构可帮助学生编写Shell脚本以自定义和自动化计算环境。

通过完成本实验，掌握下列技能：

列出编写shell脚本时使用的常见结构

创建和执行简单的shell脚本

2. 实验环境

2.1 软件环境：

1. CentOS Linux release 7.9.2009
2. VMware Workstation Pro 15
3. MobaXterm v20.0 2.2

2.2 硬件环境：

1. 阿里云云服务器ECS，共享计算型实例，配置（CPU 1核 \ 内存 2GB \ 带宽 1Mbps）
2. 个人笔记本，配置（建议内存>4G)

3. 实验步骤和方法

3.1 使用位置参数

1. 打开终端软件ssh会话，连接云服务器ECS，使用账号和密码登录。

   > 备注：如果在本地虚拟机的图形环境中实验，请以tux1身份登录，并打开终端仿真器

2. 在bin目录中，创建一个名为parameters的shell脚本，该脚本将回显以下内容：

   • Shell脚本的名称

- 前三个位置参数
- 位置参数总数

使用位置参数10 100 1000执行脚本。

```
[tux1@localhost ~]$ cd ~/bin
[tux1@localhost bin]$ vi parameters_
```

```
echo The name of this shell script is $0.
echo The first parameter passed is number $1.
echo The second parameter passed is number $2.
echo The third parameter passed is number $3.
echo Altogether there were $# parameters passed.
~
~
```

```
[tux1@localhost bin]$ chmod +x parameters
[tux1@localhost bin]$ parameters 10 100 1000
The name of this shell script is /home/tux1/bin/parameters.
The first parameter passed is number 10.
The second parameter passed is number 100.
The third parameter passed is number 1000.
Altogether there were 3 parameters passed.
[tux1@localhost bin]$
```

3. 现在，使用位置参数“10 100 1000”执行脚本，有什么区别?

```
[tux1@localhost bin]$ chmod +x parameters
[tux1@localhost bin]$ parameters 10 100 1000
The name of this shell script is /home/tux1/bin/parameters.
The first parameter passed is number 10.
The second parameter passed is number 100.
The third parameter passed is number 1000.
Altogether there were 3 parameters passed.
[tux1@localhost bin]$ parameters "10 100 1000"
The name of this shell script is /home/tux1/bin/parameters.
The first parameter passed is number 10 100 1000.
The second parameter passed is number .
The third parameter passed is number .
Altogether there were 1 parameters passed.
[tux1@localhost bin]$ _
```

3.2 条件执行

1. 使用条件执行，创建一个名为checkfile的shell脚本，该脚本检查目录中是否存在名为parameters 的文件。如果存在，请使用命令显示文件内容。执行脚本。

```
[ -f parameters ] && cat parameters
```

```
[tux1@localhost bin]$ chmod +x checkfile
[tux1@localhost bin]$ ./checkfilde
-bash: ./checkfilde: No such file or directory
[tux1@localhost bin]$ ./checkfile
echo The name of this shell script is $0.
echo The first parameter passed is number $1.
echo The second parameter passed is number $2.
echo The third parameter passed is number $3.
echo Altogether there were $# parameters passed.
[tux1@localhost bin]$ _
```

2. 修改checkfile脚本，并将所查找文件名从parameters更改为noname（检查以确保当前目录中没 有该名称的文件）。另外，使用条件执行时，如果cat命令未成功，则显示错误消息“找不到文件”。 执行脚本。

```
[tux1@localhost bin]$ chmod +x checkfile
[tux1@localhost bin]$ ./checkfilde
-bash: ./checkfilde: No such file or directory
[tux1@localhost bin]$ ./checkfile
echo The name of this shell script is $0.
echo The first parameter passed is number $1.
echo The second parameter passed is number $2.
echo The third parameter passed is number $3.
echo Altogether there were $# parameters passed.
[tux1@localhost bin]$ _
```

3. 修改checkfile脚本，以接受来自命令行的单个参数作为ls和cat命令的输入。执行脚本两次，一次使 用名为parameters的文件，再一次使用noname的文件。

```
"checkfile" 1L, 45C written
[tux1@localhost bin]$ ./checkfile parameters
echo The name of this shell script is $0.
echo The first parameter passed is number $1.
echo The second parameter passed is number $2.
echo The third parameter passed is number $3.
echo Altogether there were $# parameters passed.
[tux1@localhost bin]$ ./checkfile noname
noname was not found
[tux1@localhost bin]$
```

4. 再次执行checkfile脚本，但是这次不使用任何参数。会发生什么？修改脚本，这样就不会再次发生。

```
"checkfile" 1L, 47C written
[tux1@localhost bin]$ ./checkfile
abc
abc
```

3.3 循环

1. 使用for循环，修改checkfile脚本以接受多个文件作为命令行输入，而不仅仅是一个。如果找到文 件，则显示文件内容。如果找不到文件，则显示一条错误消息，显示未找到的所有文件名。在目录 中查找并记下一些可用作输入的有效文件名。使用有效和无效的文件名执行脚本。

```
for x in "$@"
do
[ -f "$x" ] && cat "$x" || echo "$x was not found"
done
```

```
"checkfile" 4L, 73C written
[tux1@localhost bin]$ ./checkfile filename filename filename
filename was not found
filename was not found
filename was not found
[tux1@localhost bin]$ ./checkfile parameters parameters parameters
echo The name of this shell script is $0.
echo The first parameter passed is number $1.
echo The second parameter passed is number $2.
echo The third parameter passed is number $3.
echo Altogether there were $# parameters passed.
echo The name of this shell script is $0.
echo The first parameter passed is number $1.
echo The second parameter passed is number $2.
echo The third parameter passed is number $3.
echo Altogether there were $# parameters passed.
echo The name of this shell script is $0.
echo The first parameter passed is number $1.
echo The second parameter passed is number $2.
echo The third parameter passed is number $3.
echo Altogether there were $# parameters passed.
[tux1@localhost bin]$
```

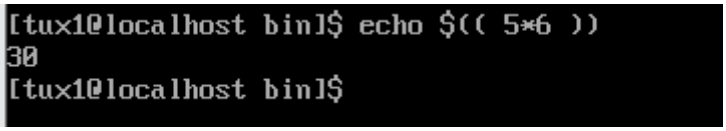
2. 现在做同样的事情，但是结合使用while循环和shift命令。

```
while [ ! -z "$1" ]
do
[ -f "$1" ] && cat "$1" || echo "$1 was not found"
shift
done
~
```

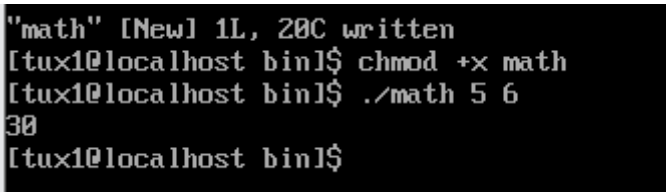
```
"checkfile" 5L, 85C written
[tux1@localhost bin]$ ./checkfile filename filename filename
filename was not found
filename was not found
filename was not found
[tux1@localhost bin]$ ./checkfile parameters parameters parameters
echo The name of this shell script is $0.
echo The first parameter passed is number $1.
echo The second parameter passed is number $2.
echo The third parameter passed is number $3.
echo Altogether there were $# parameters passed.
echo The name of this shell script is $0.
echo The first parameter passed is number $1.
echo The second parameter passed is number $2.
echo The third parameter passed is number $3.
echo Altogether there were $# parameters passed.
echo The name of this shell script is $0.
echo The first parameter passed is number $1.
echo The second parameter passed is number $2.
echo The third parameter passed is number $3.
echo Altogether there were $# parameters passed.
[tux1@localhost bin]$ _
```

3.4 算术

1. 从命令行显示乘以5乘以6的结果。



2. 现在，创建一个名为math的shell脚本，当从命令行作为输入输入时，将任意两个数字相乘。执行 脚本5乘以6。尝试其他任意两个数字。

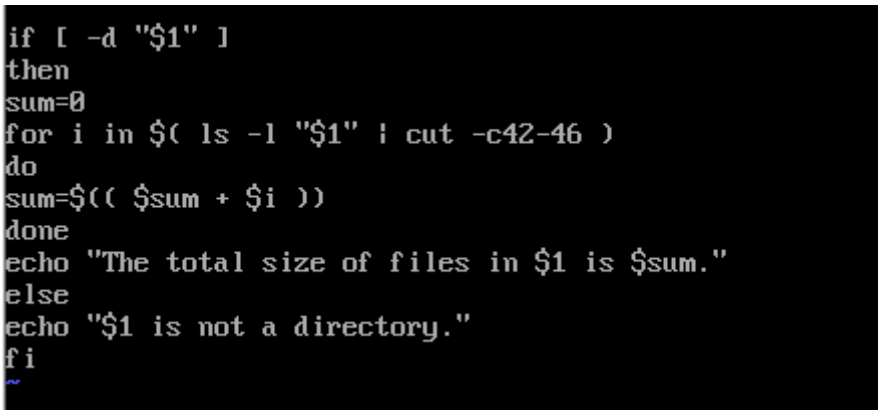


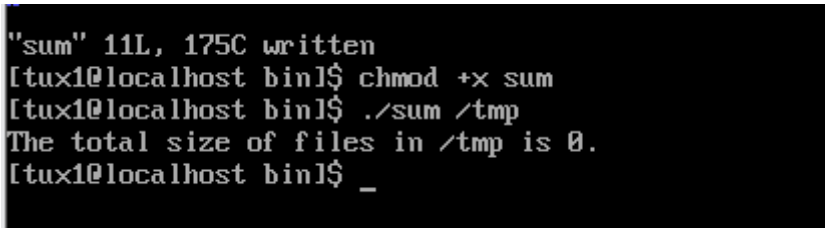
3.5 整合练习

1. 使用您在本课程中获得的知识编写一个脚本，该脚本接受目录名称作为参数，并计算该目录中文件 的总大小。

> 笔记：
>
> 根据发行版中ls命令的默认值，cut命令的列号可能需要稍作调整，而且不同长度的用户名和文件大小也会影响取字符的位置。
>
> 实际上，使用上述的cut命令并不会获得真正可移植的shell脚本，因为ls -l根据列中数据的宽度来 格式化其列。同时，cut会将一系列的多个空格（用作填充字符以正确对齐列）用作多个分隔符， 因此您也不能使用cut -d’ ’ -f 5
>
> 这是一个效果更好的示例：

> ls -l | awk '{print $ 5}'
>
> awk在“Shells基础”单元中进行了简要介绍，并将在“ Shell编程”课程中进行深入介绍。





实验4.3 搭建shell脚本开发环境（VSCode）

1. 实验目的

让学生能够安装VSCode和相关插件，搭建一个跨平台的shell脚本开发环境。通过完成本实验，掌握下 列技能：使用VSCode+Remote-SSH远程编写shell脚本 使用VSCode+BASH debug调试shell脚本

2. 实验环境

2.1 软件环境：

1. CentOS Linux release 7.9.2009
2. VMware Workstation Pro 15
3. MobaXterm v20.0 2.2

2.2 硬件环境：

1. 阿里云云服务器ECS，共享计算型实例，配置（CPU 1核 \ 内存 2GB \ 带宽 1Mbps）
2. 个人笔记本，配置（建议内存>4G）

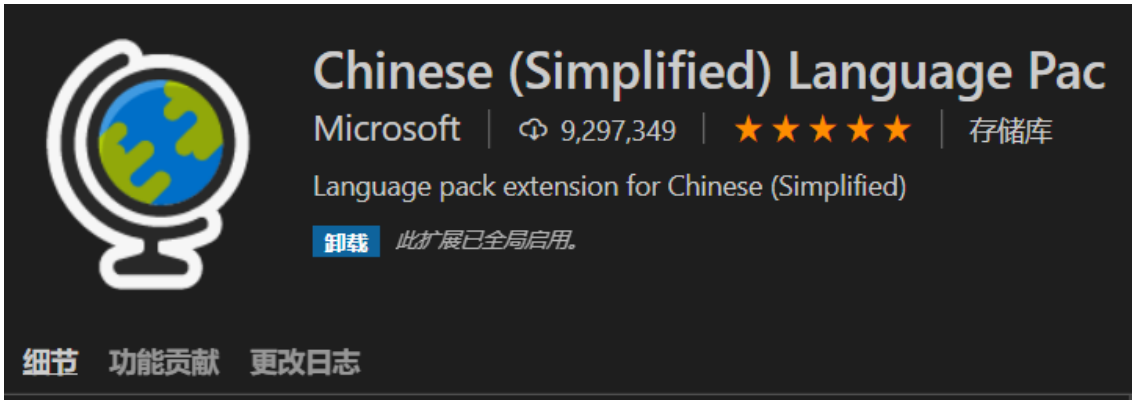
3. 实验步骤和方法

3.1 VS Code配置Remote-SSH远程开发

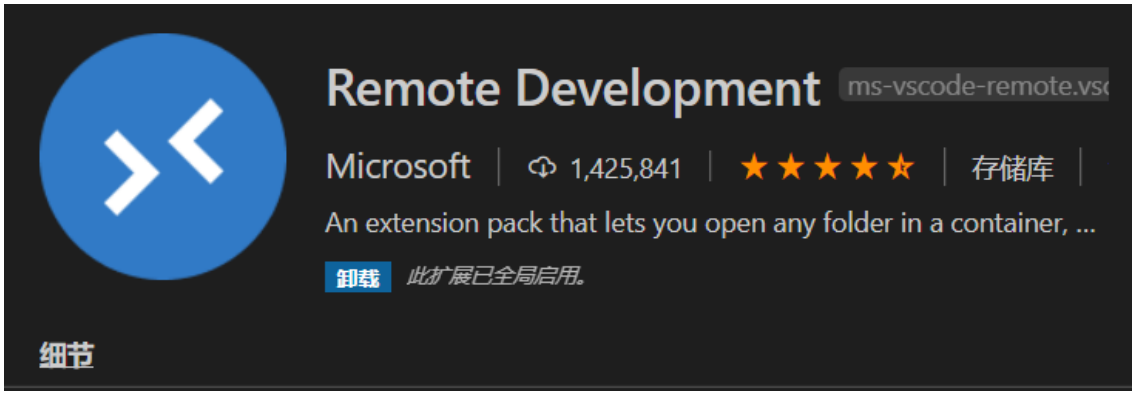
1. 下载并安装VS Code

   官网下载链接：https://code.visualstudio.com/

   提示：安装后默认是英文界面，要改成中文的话可以点击左侧的扩展按钮（Ctrl+Shift+x）搜 索“Chinese”，安装搜索结果中的第一个（Chinese (Simplified) Language Pack for Visual Studio Code）后重启软件即可变为中文。



2. 安装远程开发扩展 点击扩展栏，然后搜索“Remote Development”，第一个结果就是我们需要的插件，点击安装。

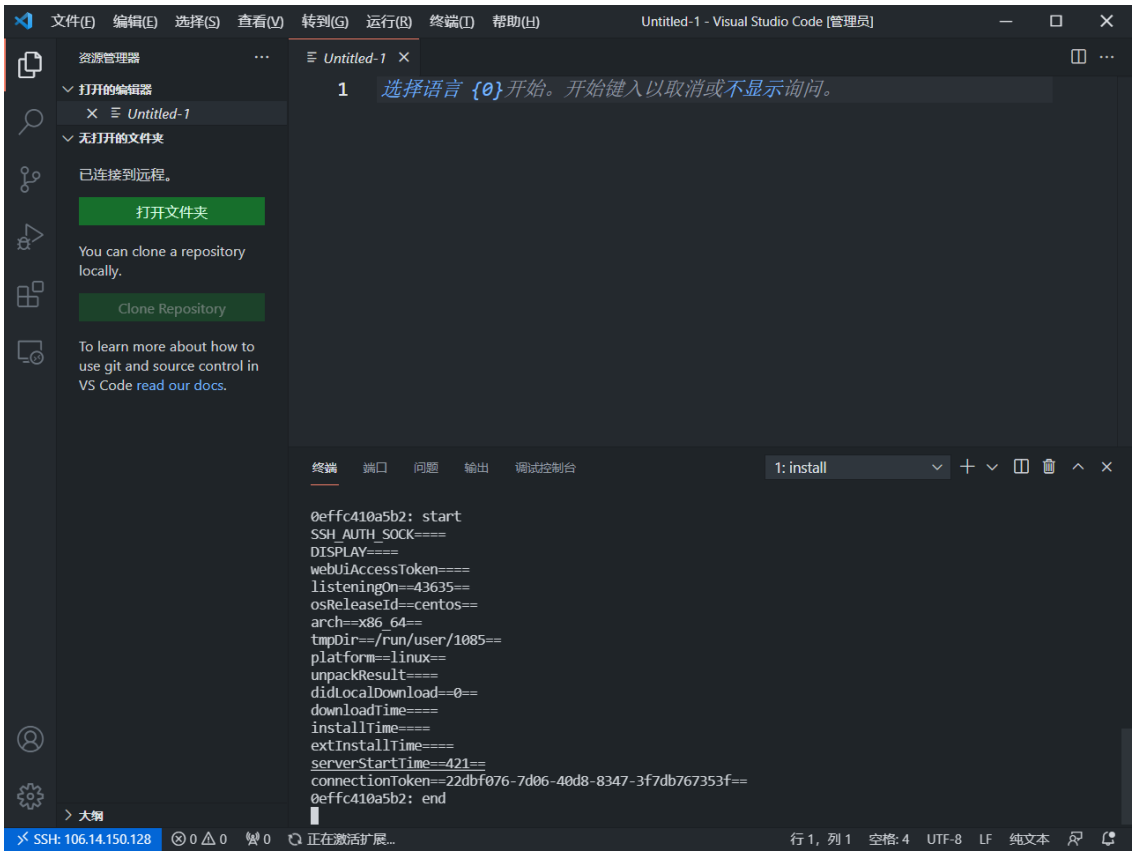


3. 安装SSH工具

实现远程连接服务器需要SSH客户端，Linux 系统一般自带 SSH 工具，但是 Windows 下就需要自 行安装。推荐 Windows 版本 Git 自带的 SSH，下载链接：https://git-scm.com/download/win ， 选择系统对应的版本，安装版本（Setup）会自动配置好环境变量，无需其他操作；绿色版本 （Protable）则需要手动添加环境变量。安装好后可以打开命令提示符（cmd），输入 ssh 验证一下，出现如下命令回显即为正常。

4. 配置Remote-SSH

打开VS Code，点击右侧远程资源管理器，选择SSH Targets。点击add New，输入ssh 用户名@IP （例：ssh tux1@106.14.150.128）。选择配置文件，默认选择第一个。点击新添加的远程服务器，在当前窗口打开，输入密码。左下角出现如下提示表示连接成功。

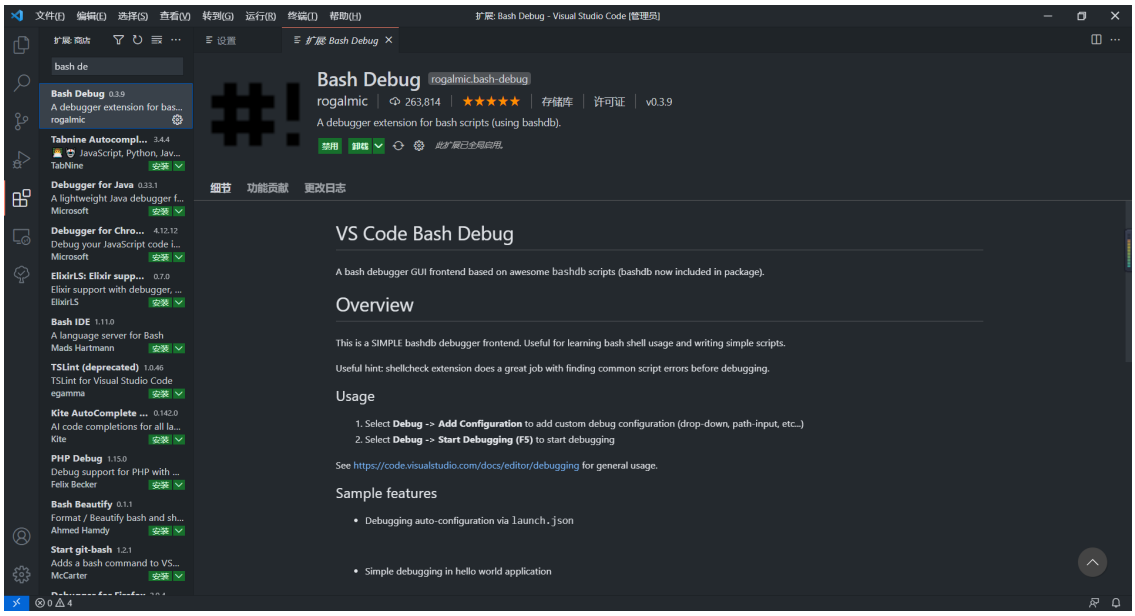


> 提示：上述配置方法在每次连接时都需要输入密码，还有另外一种通过密钥登录的方式。 密钥登录教程：打开 cmd 或 git bash，输入 ssh-keygen 后一路回车即可。这会在你用户目 录下的 .ssh 文件夹内生成 id_rsa 和 id_rsa.pub 两个文件，分别对应为私钥和公钥。将 id_rsa.pub 文件中的内容复制到你远程主机用户目录下 .ssh 文件夹内名为 authorized_keys 的文件中即可。

3.2 VS Code Bash debug

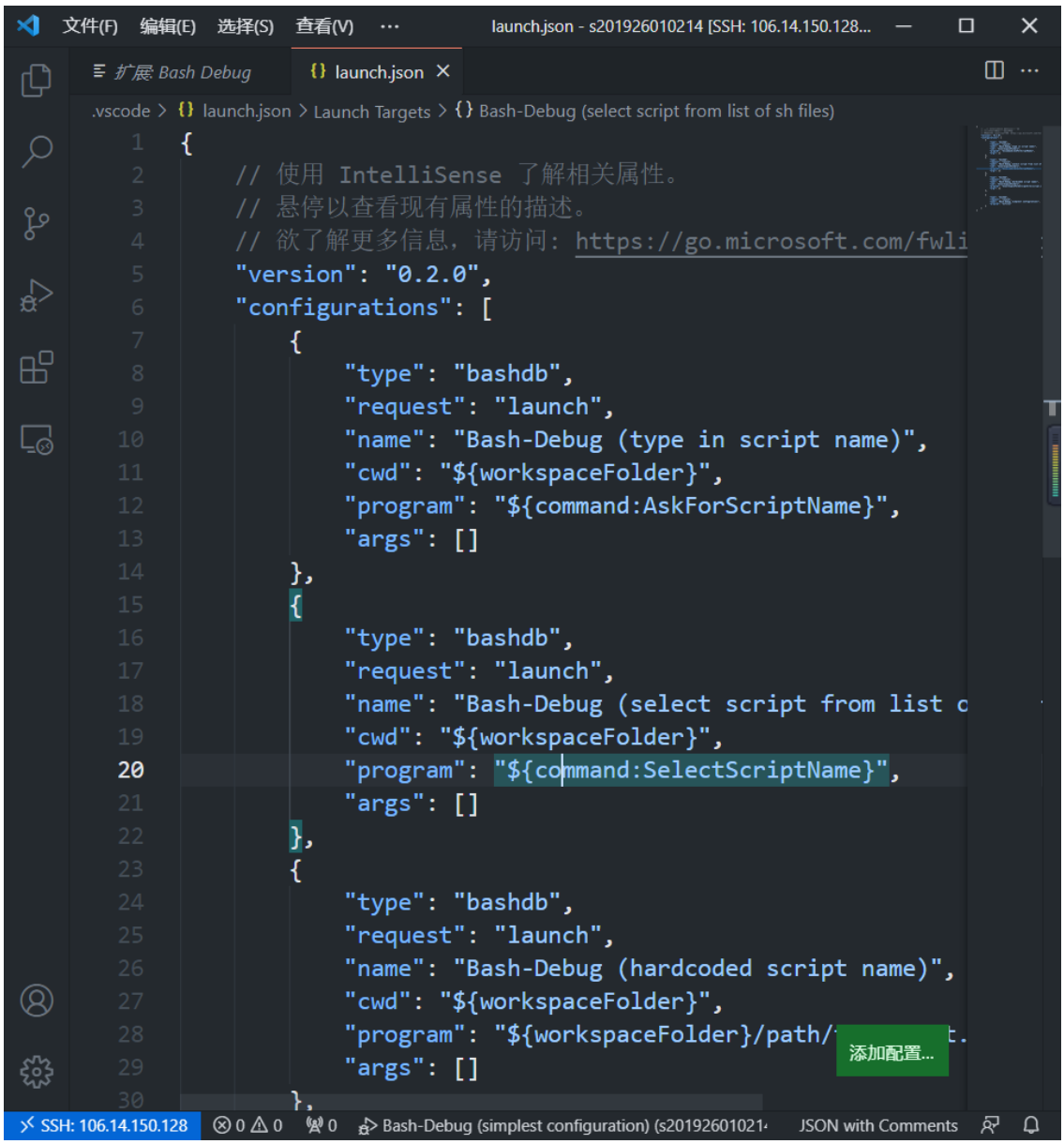
1. 安装Bash Debug

   点击扩展栏，然后搜索“Bash Debug”，点击安装。



2. 添加调试配置文件（可以在本地或云服务上实现）。

   确保SSH连接成功，若在本地进行可按照之前的方式连接至虚拟机。点击"文件->打开文件夹"，选 择对应的用户文件夹。点击"运行->添加配置"，系统会自动生成.vscode文件夹和launch.json文 件，在launch.json中添加以下三个部分，并保存。



3. 创建测试文件

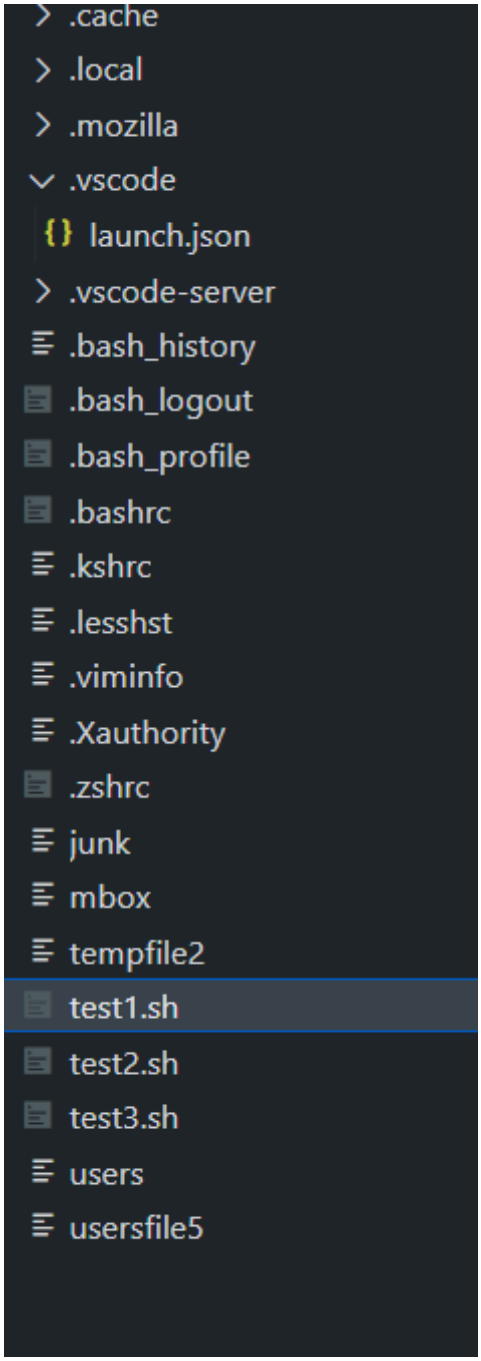
创建test1.sh、test2.sh、test3.sh并在文件中输入以下内容：

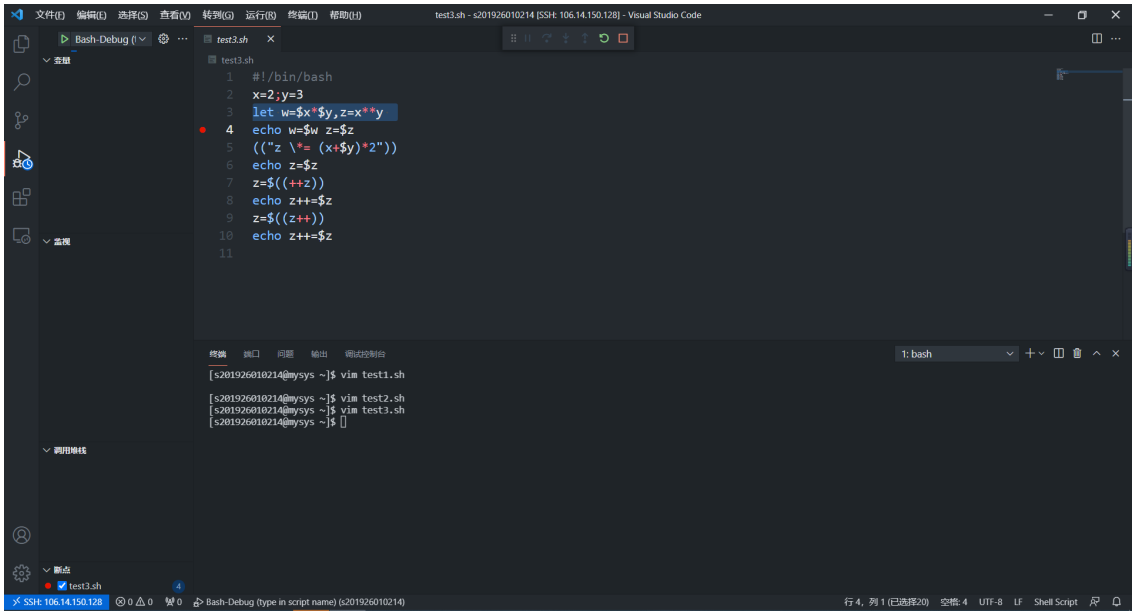
```
$ vim test1.sh
echo this is test1!
source test2.sh

$ vim test2.sh
echo this is test2!

$ vim test3.sh
```

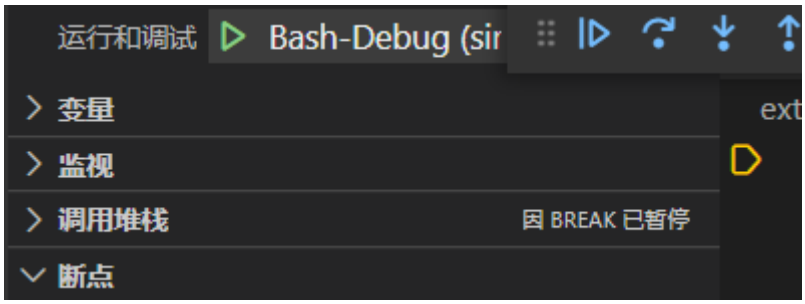
```
#!/bin/bash
x=2;y=3
let w=$x*$y,z=x**y
echo w=$w z=$z
(("z \*= (x+$y)*2"))
echo z=$z
z=$((++z))
echo z++=$z
z=$((z++))
echo z++=$z
```





4. 利用调试工具栏，对脚本进行简单调试

a. 打开test1.sh，点击左边栏“运行和调试”或者按热键，启动调试，选择Bash-Debug(simplest configuration) 点击开始调试。



b. 点击查看，点击打开调试控制台，在调试工具栏中进行单步调试，观察调试控制台的输出。

c. 调试控制台输出结果

```
this is test2!
```

5. 利用调试工具栏，对脚本进行复杂调试

a. 在test3.sh第9行设置断点

b. 添加监视变量值

c. 调试脚本，（F11)单步调试test3.sh，观察变量、监视和调用堆栈的变化