

Processes

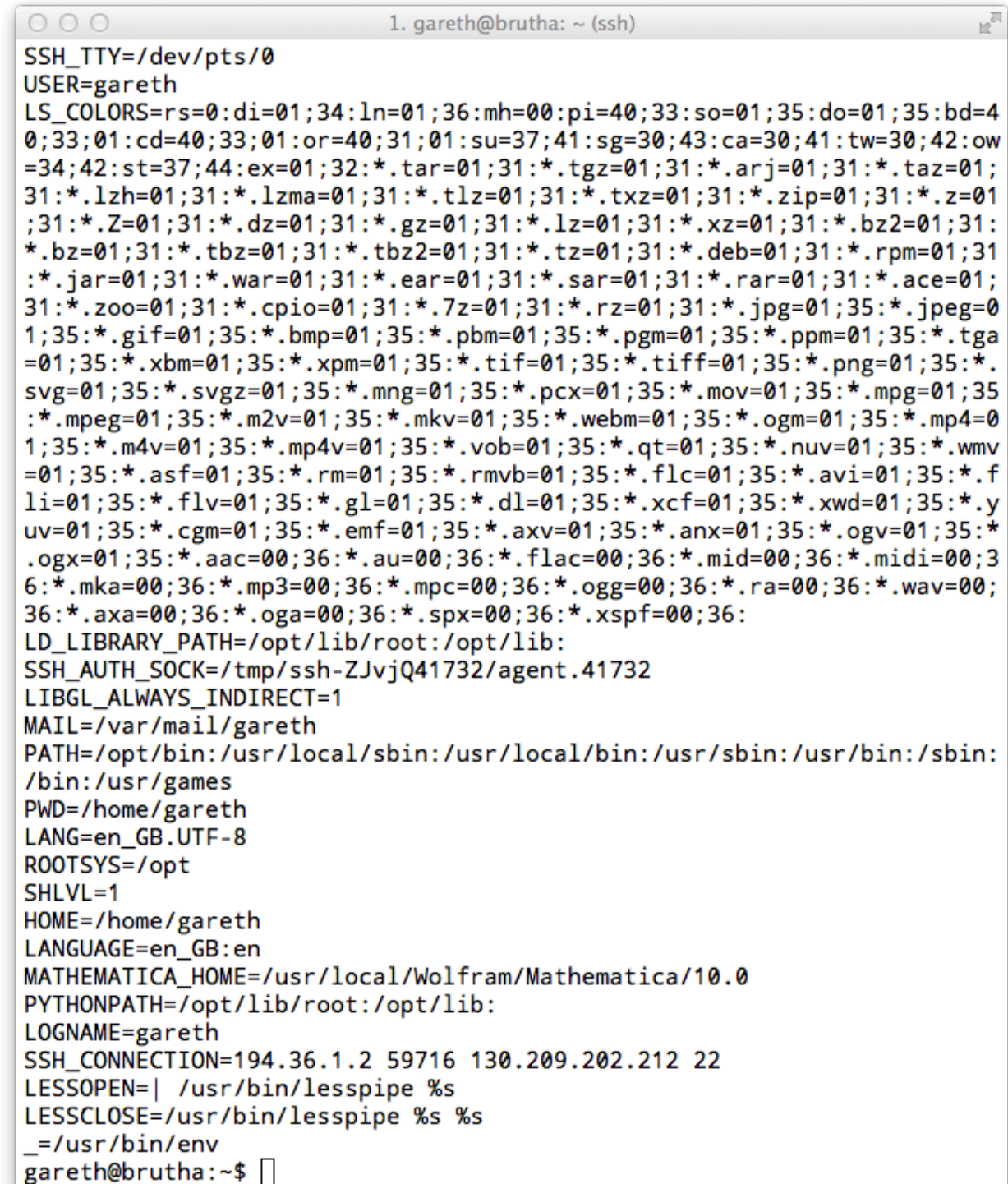
Dr. Gareth Roy (x6439)
gareth.roy@glasgow.ac.uk

- Variables and the Shell
- Processes
- Working with Processes
- IO Streams
- Pipes and Redirections

- Variables and the Shell
- Processes
- Working with Processes
- IO Streams
- Pipes and Redirections

Variables

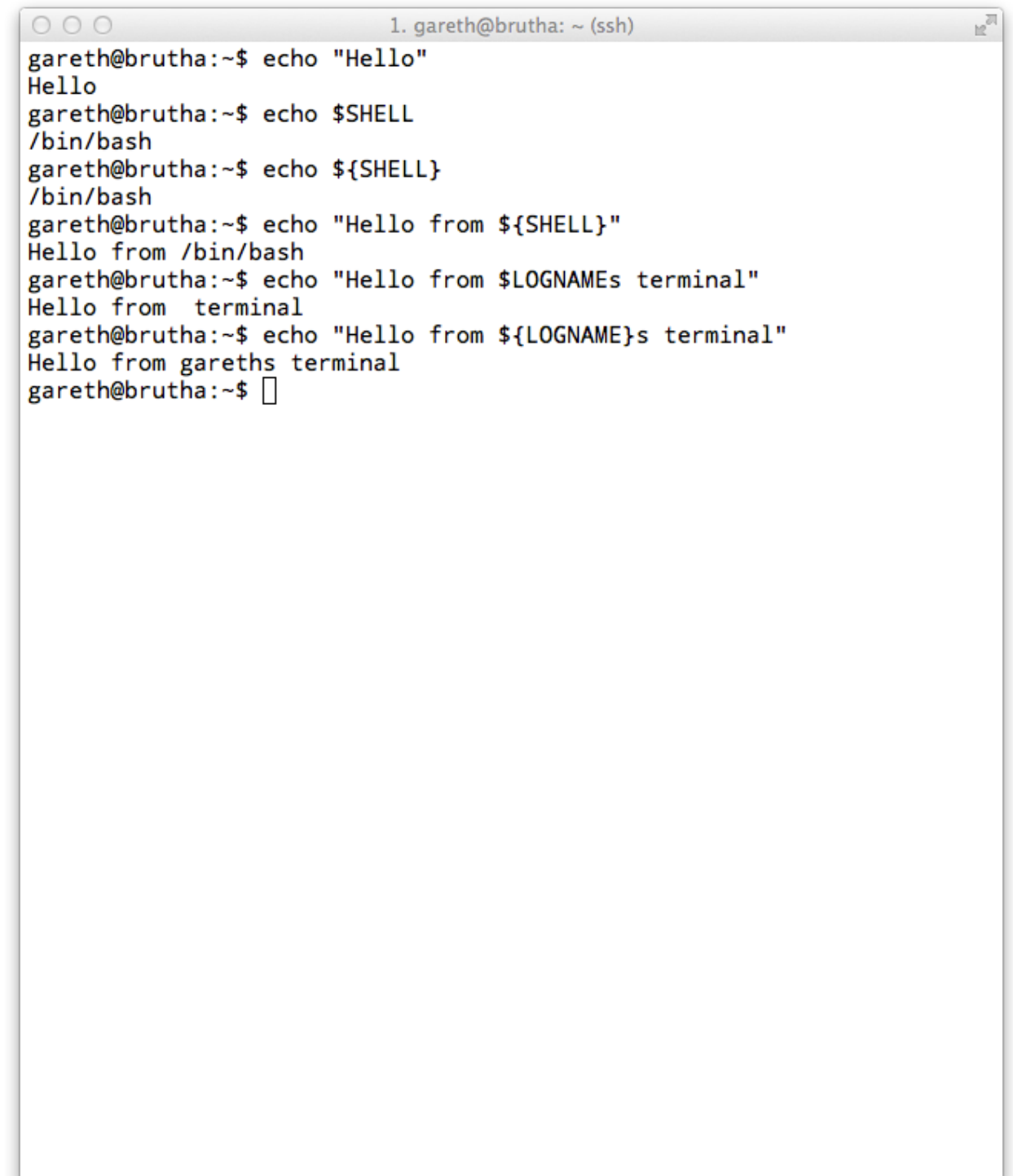
- The “shell” stores parameters in variables. These are referred to as environment variables.
- In these examples we assume our shell is **bash**.
- To assign a value to a bash variable on the command line we would do:
 - **export** SOME_VAR=“some value”
 - **Note:** there are no spaces between the “=” and the left and right hand side of the assignment and variables are case sensitive.
- **bash** and most shells use these variables to store configuration information.
- By convention most environment variable names are all in uppercase.
- You can see all the variables set in a running shell by using the **env** or **export** commands.

A terminal window titled '1. gareth@brutha: ~ (ssh)' showing the output of the 'env' command. The output lists various environment variables such as SSH_TTY, USER, LS_COLORS, PATH, PWD, and more. The prompt at the bottom is 'gareth@brutha:~\$'.

```
SSH_TTY=/dev/pts/0
USER=gareth
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=4
0;33;01:cd=40;33;01:or=40;31;01:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow
=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;
31:*.lzh=01;31:*.lzm=01;31:*.tlz=01;31:*.txz=01;31:*.zip=01;31:*.z=01
;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lz=01;31:*.xz=01;31:*.bz2=01;31:
*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31
:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.ace=01;
31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.jpg=01;35:*.jpeg=0
1;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga
=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.
svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35
:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=0
1;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv
=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.f
li=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.y
uv=01;35:*.cgm=01;35:*.emf=01;35:*.axv=01;35:*.anx=01;35:*.ogv=01;35:*.
ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.mid=00;36:*.midi=00;3
6:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;
36:*.axa=00;36:*.oga=00;36:*.spx=00;36:*.xspf=00;36:
LD_LIBRARY_PATH=/opt/lib/root:/opt/lib:
SSH_AUTH_SOCK=/tmp/ssh-ZJvjQ41732/agent.41732
LIBGL_ALWAYS_INDIRECT=1
MAIL=/var/mail/gareth
PATH=/opt/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
/bin:/usr/games
PWD=/home/gareth
LANG=en_GB.UTF-8
ROOTSYS=/opt
SHLVL=1
HOME=/home/gareth
LANGUAGE=en_GB:en
MATHEMATICA_HOME=/usr/local/Wolfram/Mathematica/10.0
PYTHONPATH=/opt/lib/root:/opt/lib:
LOGNAME=gareth
SSH_CONNECTION=194.36.1.2 59716 130.209.202.212 22
LESSOPEN=| /usr/bin/lesspipe %s
LESSCLOSE=/usr/bin/lesspipe %s %s
_=/usr/bin/env
gareth@brutha:~$
```

Variables

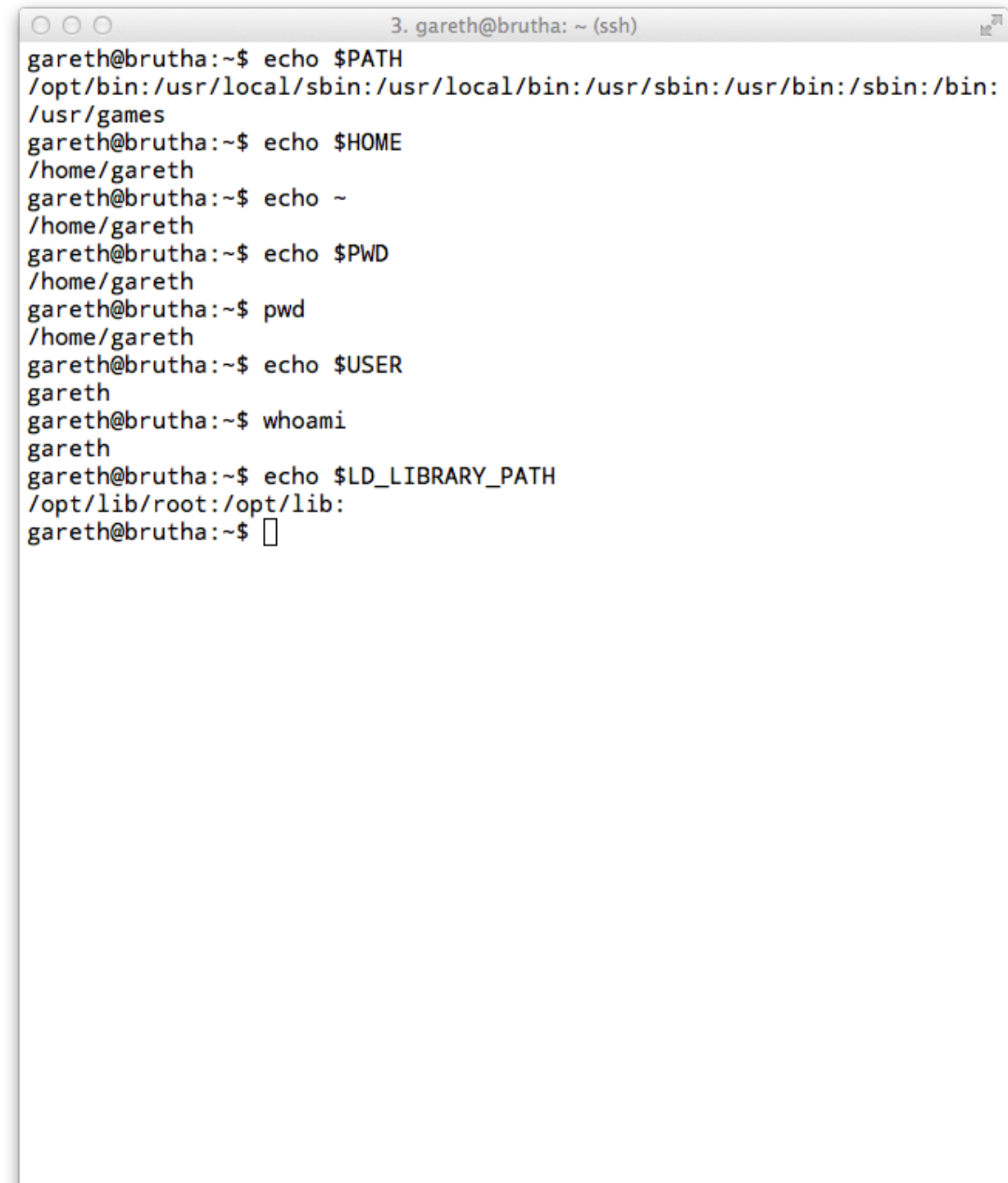
- To access a specific variable we can use the **echo** command.
- **echo** writes to the screen anything passed to it:
 - **echo** "hello"
- To access a specific shell variable we can echo it's contents by adding **\$** to its name:
 - **echo** \$SHELL
- It's better practice to reference our variables using **\${}** notation. The above would then be
 - **echo** \${SHELL}
- This allows us to embed the variable expansion inside another value without worrying about the variable name:
 - **echo** "Hello from \${SHELL}"



```
1. gareth@brutha: ~ (ssh)
gareth@brutha:~$ echo "Hello"
Hello
gareth@brutha:~$ echo $SHELL
/bin/bash
gareth@brutha:~$ echo ${SHELL}
/bin/bash
gareth@brutha:~$ echo "Hello from ${SHELL}"
Hello from /bin/bash
gareth@brutha:~$ echo "Hello from $LOGNAME's terminal"
Hello from terminal
gareth@brutha:~$ echo "Hello from ${LOGNAME}'s terminal"
Hello from gareth's terminal
gareth@brutha:~$
```

Important Variables

- There are some shell variables that it is useful to know about
- **\$PATH** - the places to look for a particular command.
- **\$HOME** - the user's home directory (can be accessed using the `~` symbol)
- **\$PWD** - the current working directory (also available via **pwd** command)
- **\$USER** - the username (also available via the **whoami** command)
- **\$LD_LIBRARY_PATH** - the paths to search for library files when building C code.

A terminal window titled "3. gareth@brutha: ~ (ssh)" showing the output of several shell commands. The commands and their outputs are: `echo $PATH` returns `/opt/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games`; `echo $HOME` returns `/home/gareth`; `echo ~` returns `/home/gareth`; `echo $PWD` returns `/home/gareth`; `pwd` returns `/home/gareth`; `echo $USER` returns `gareth`; `whoami` returns `gareth`; and `echo $LD_LIBRARY_PATH` returns `/opt/lib/root:/opt/lib:`.

```
gareth@brutha:~$ echo $PATH
/opt/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
gareth@brutha:~$ echo $HOME
/home/gareth
gareth@brutha:~$ echo ~
/home/gareth
gareth@brutha:~$ echo $PWD
/home/gareth
gareth@brutha:~$ pwd
/home/gareth
gareth@brutha:~$ echo $USER
gareth
gareth@brutha:~$ whoami
gareth
gareth@brutha:~$ echo $LD_LIBRARY_PATH
/opt/lib/root:/opt/lib:
gareth@brutha:~$
```


Aliases

- Often typed commands can be “aliased” to much simpler commands.
- For instance if we would like a long directory listing, showing all hidden files and a classifier:

- **ls -laF**

- This could be aliased to

- **alias ll='ls -laF'**

- At the command line we would then use:

- **ll**

- You can see a list of all aliases set in your shell by running the alias command without any arguments

- **alias**

```
3. gareth@brutha: ~/Examples (ssh)
gareth@brutha:~/Examples$ ls -laF
total 12
drwxrwxr-x  3 gareth gareth 4096 Jan 21 13:22 ./
drwxr-x--- 29 gareth gareth 4096 Jan 20 16:23 ../
-r--r--r--  1 gareth gareth   0 Jan 14 09:05 a
-rw-rw-r--  1 gareth gareth   0 Jan 14 09:05 b
-rw-rw-r--  1 gareth gareth   0 Jan 14 13:59 c
drwxrwxr-x  4 gareth gareth 4096 Jan 14 09:28 dir1/
gareth@brutha:~/Examples$ alias ll='ls -laF'
gareth@brutha:~/Examples$ ll
total 12
drwxrwxr-x  3 gareth gareth 4096 Jan 21 13:22 ./
drwxr-x--- 29 gareth gareth 4096 Jan 20 16:23 ../
-r--r--r--  1 gareth gareth   0 Jan 14 09:05 a
-rw-rw-r--  1 gareth gareth   0 Jan 14 09:05 b
-rw-rw-r--  1 gareth gareth   0 Jan 14 13:59 c
drwxrwxr-x  4 gareth gareth 4096 Jan 14 09:28 dir1/
gareth@brutha:~/Examples$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo termin
al || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s
/[\;&]\s*alert$//'\`)'"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -laF'
alias ls='ls --color=auto'
gareth@brutha:~/Examples$
```

Persistent Variables

- All variables in **bash** shells are transient.
- Variables you set in one shell will not persist when you open a new one.
- When you open a new shell bash reads a number of configuration files to set its state.
- We can make variables persist by adding them to one of these configuration files.
- Bash has two configuration files found in your home directory
 - **.bashrc**
 - **.bash_profile**
- The difference is subtle and most times you'll want to put settings in **.bashrc** but be aware that **.bash_profile** exists.
- If you edit your **.bashrc** file you can load the new variables into the current shell by
 - **source .bashrc**

```
3. gareth@brutha: ~ (ssh)
gareth@brutha:~$ export HELLO_WORLD="Hello"
gareth@brutha:~$ echo ${HELLO_WORLD}
Hello
gareth@brutha:~$ exit
logout
Connection to 130.209.202.212 closed.
Folkvangr:Desktop gareth$ ssh brutha
Welcome to Ubuntu 12.04.5 LTS (GNU/Linux 3.13.0-43-generic x86_64)

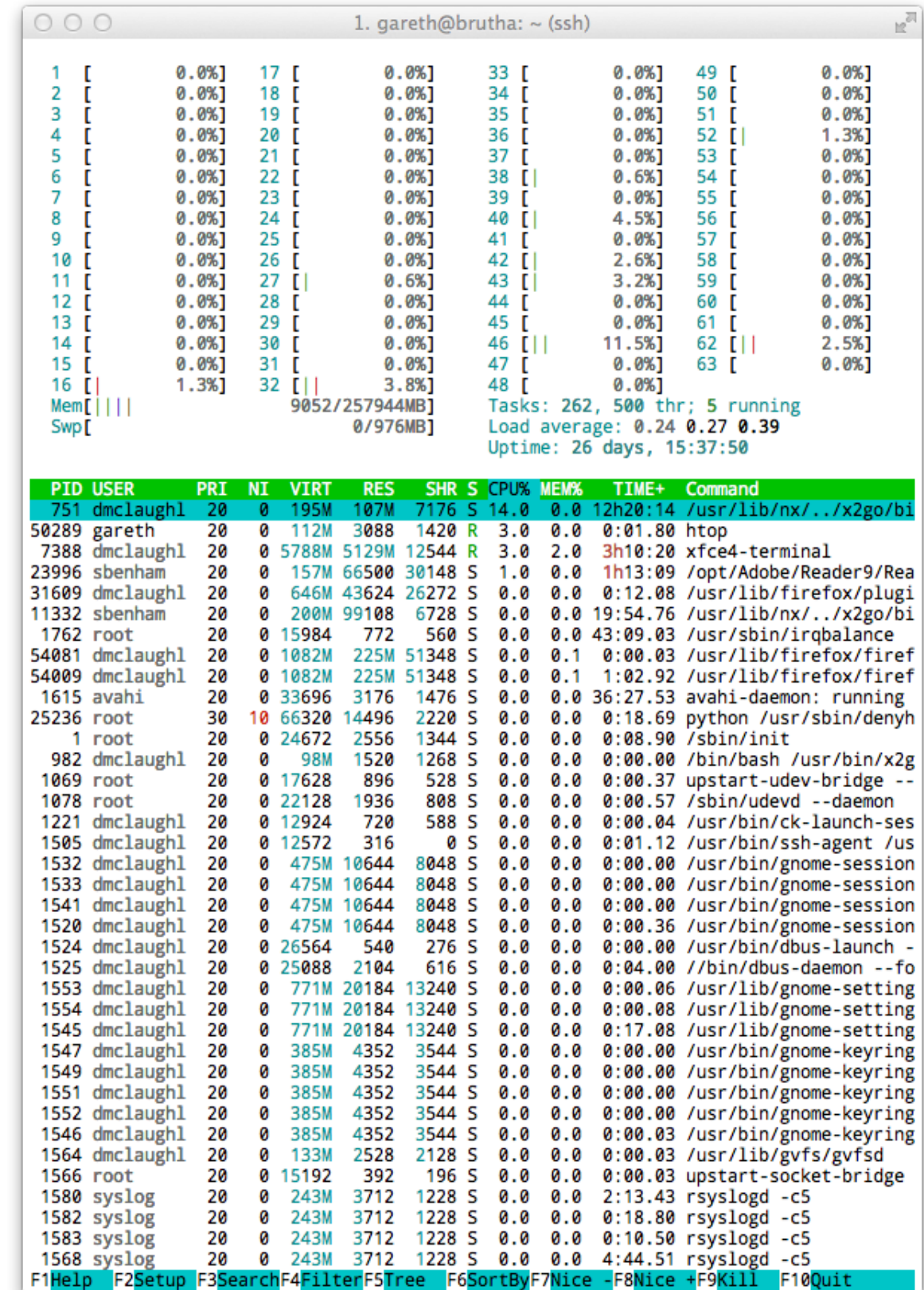
 * Documentation:  https://help.ubuntu.com/
Last login: Wed Jan 21 13:29:37 2015 from ppegw.physics.gla.ac.uk
gareth@brutha:~$ echo ${HELLO_WORLD}

gareth@brutha:~$ grep alias .bashrc
# enable color support of ls and also add handy aliases
alias ls='ls --color=auto'
#alias dir='dir --color=auto'
#alias vdir='vdir --color=auto'
alias grep='grep --color=auto'
alias fgrep='fgrep --color=auto'
alias egrep='egrep --color=auto'
# some more ls aliases
alias ll='ls -aF'
alias la='ls -A'
alias l='ls -CF'
# Add an "alert" alias for long running commands. Use like so:
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo termin
al || echo error)" "${history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s
/[\;&|]\s*alert$/\s*\''}"'
# ~/.bash_aliases, instead of adding them here directly.
if [ -f ~/.bash_aliases ]; then
. ~/.bash_aliases
gareth@brutha:~$
```


- Variables and the Shell
- Processes
- Working with Processes
- IO Streams
- Pipes and Redirections

What is a Process?

- Each program that runs on a Linux system runs inside a process.
- Each process is self contained, even when it's for the same program.
- The Linux Kernel schedules CPU time for each process to run.
- Processes are created as children of other processes by a method called “**forking**”.
- **ps** will show all the processes running on a system, and the relationship they have with each other.



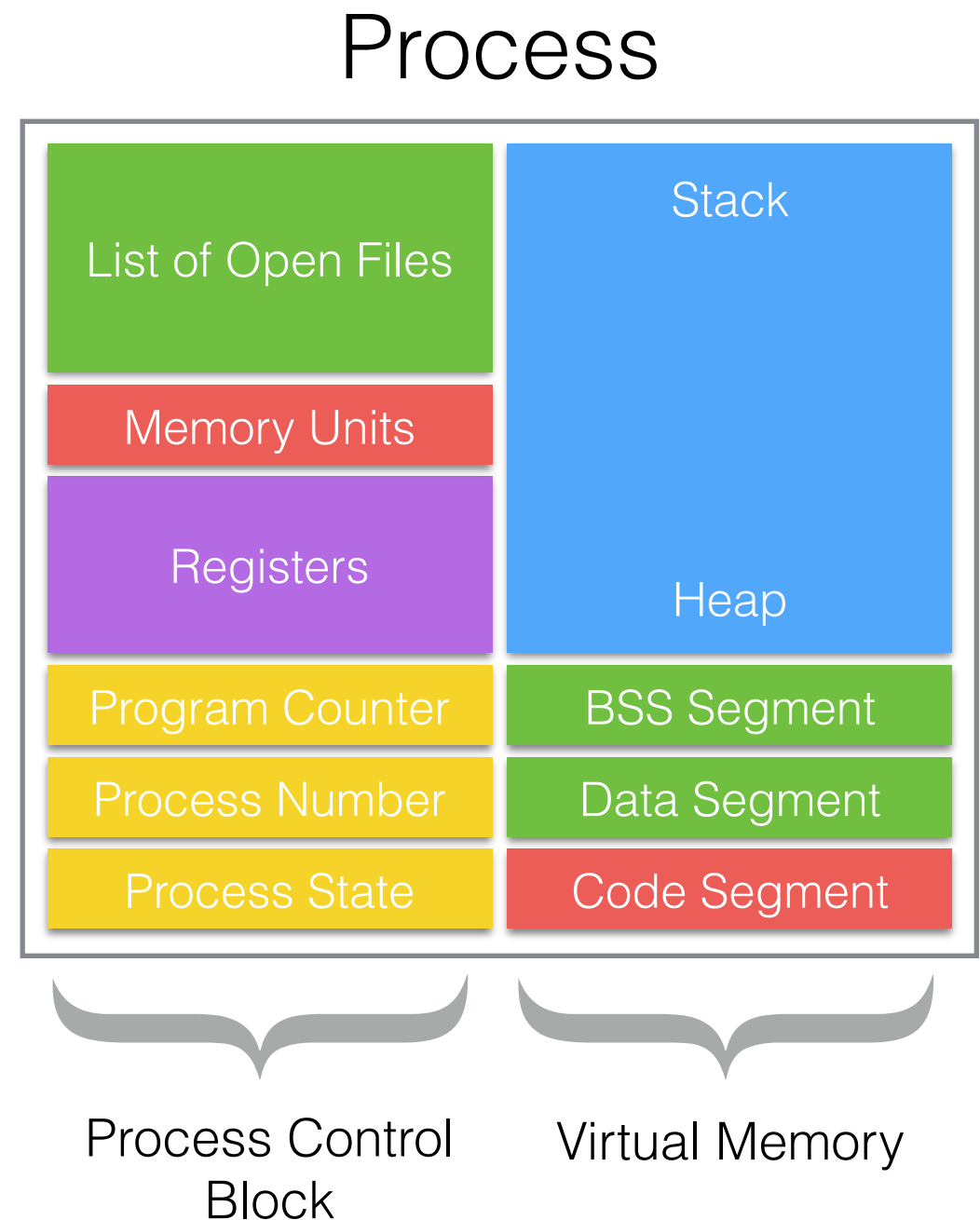
```
1. gareth@brutha: ~ (ssh)
```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
751	dmcloughl	20	0	195M	107M	7176	S	14.0	0.0	12h20:14	/usr/lib/nx/./x2go/bi
50289	gareth	20	0	112M	3088	1420	R	3.0	0.0	0:01.80	htop
7388	dmcloughl	20	0	5788M	5129M	12544	R	3.0	2.0	3h10:20	xfce4-terminal
23996	sbenham	20	0	157M	66500	30148	S	1.0	0.0	1h13:09	/opt/Adobe/Reader9/Rea
31609	dmcloughl	20	0	646M	43624	26272	S	0.0	0.0	0:12.08	/usr/lib/firefox/plugi
11332	sbenham	20	0	200M	99108	6728	S	0.0	0.0	19:54.76	/usr/lib/nx/./x2go/bi
1762	root	20	0	15984	772	560	S	0.0	0.0	43:09.03	/usr/sbin/irqbalance
54081	dmcloughl	20	0	1082M	225M	51348	S	0.0	0.1	0:00.03	/usr/lib/firefox/firef
54009	dmcloughl	20	0	1082M	225M	51348	S	0.0	0.1	1:02.92	/usr/lib/firefox/firef
1615	avahi	20	0	33696	3176	1476	S	0.0	0.0	36:27.53	avahi-daemon: running
25236	root	30	10	66320	14496	2220	S	0.0	0.0	0:18.69	python /usr/sbin/denyh
1	root	20	0	24672	2556	1344	S	0.0	0.0	0:08.90	/sbin/init
982	dmcloughl	20	0	98M	1520	1268	S	0.0	0.0	0:00.00	/bin/bash /usr/bin/x2g
1069	root	20	0	17628	896	528	S	0.0	0.0	0:00.37	upstart-udev-bridge --
1078	root	20	0	22128	1936	808	S	0.0	0.0	0:00.57	/sbin/udev --daemon
1221	dmcloughl	20	0	12924	720	588	S	0.0	0.0	0:00.04	/usr/bin/ck-launch-ses
1505	dmcloughl	20	0	12572	316	0	S	0.0	0.0	0:01.12	/usr/bin/ssh-agent /us
1532	dmcloughl	20	0	475M	10644	8048	S	0.0	0.0	0:00.00	/usr/bin/gnome-session
1533	dmcloughl	20	0	475M	10644	8048	S	0.0	0.0	0:00.00	/usr/bin/gnome-session
1541	dmcloughl	20	0	475M	10644	8048	S	0.0	0.0	0:00.00	/usr/bin/gnome-session
1520	dmcloughl	20	0	475M	10644	8048	S	0.0	0.0	0:00.36	/usr/bin/gnome-session
1524	dmcloughl	20	0	26564	540	276	S	0.0	0.0	0:00.00	/usr/bin/dbus-launch -
1525	dmcloughl	20	0	25088	2104	616	S	0.0	0.0	0:04.00	//bin/dbus-daemon --fo
1553	dmcloughl	20	0	771M	20184	13240	S	0.0	0.0	0:00.06	/usr/lib/gnome-setting
1554	dmcloughl	20	0	771M	20184	13240	S	0.0	0.0	0:00.08	/usr/lib/gnome-setting
1545	dmcloughl	20	0	771M	20184	13240	S	0.0	0.0	0:17.08	/usr/lib/gnome-setting
1547	dmcloughl	20	0	385M	4352	3544	S	0.0	0.0	0:00.00	/usr/bin/gnome-keyring
1549	dmcloughl	20	0	385M	4352	3544	S	0.0	0.0	0:00.00	/usr/bin/gnome-keyring
1551	dmcloughl	20	0	385M	4352	3544	S	0.0	0.0	0:00.00	/usr/bin/gnome-keyring
1552	dmcloughl	20	0	385M	4352	3544	S	0.0	0.0	0:00.00	/usr/bin/gnome-keyring
1546	dmcloughl	20	0	385M	4352	3544	S	0.0	0.0	0:00.03	/usr/bin/gnome-keyring
1564	dmcloughl	20	0	133M	2528	2128	S	0.0	0.0	0:00.03	/usr/lib/gvfs/gvfsd
1566	root	20	0	15192	392	196	S	0.0	0.0	0:00.03	upstart-socket-bridge
1580	syslog	20	0	243M	3712	1228	S	0.0	0.0	2:13.43	rsyslogd -c5
1582	syslog	20	0	243M	3712	1228	S	0.0	0.0	0:18.80	rsyslogd -c5
1583	syslog	20	0	243M	3712	1228	S	0.0	0.0	0:10.50	rsyslogd -c5
1568	syslog	20	0	243M	3712	1228	S	0.0	0.0	4:44.51	rsyslogd -c5

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit

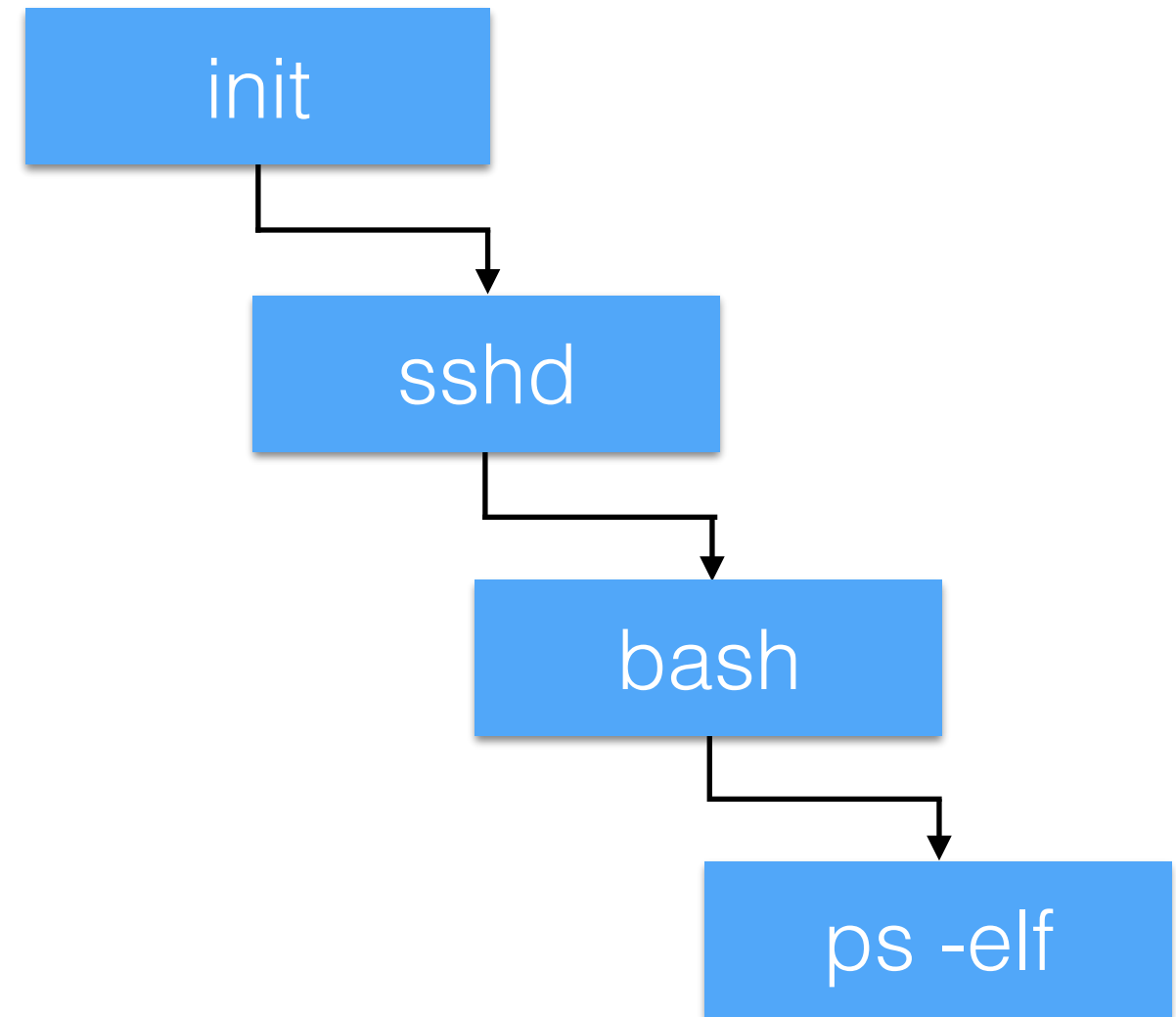
What's inside a Process?

- It contains:
 - a copy of the of the program.
 - a section of Virtual Memory assigned to the process (containing the Stack and Heap).
 - a link to any open files (file descriptors)
 - a snapshot of the processor state (including register contents)
 - attributes such as owner and permissions.
- Each process is isolated from every other process in the system.



Forking

- Processes are created by “**forking**”.
- When a process **forks** it creates a copy of itself and then starts the program it desires to run.
- This gives a parent child relationship between processes.
- The initial process is usually “**init**”



```
garth@brutha:~$ ps -elf |grep garth
4 S root      49740 38657 0 80 0 - 23088 poll_s 12:00 ?      00:00:00 sshd: garth [priv]
5 S garth     49755 49740 0 80 0 - 23088 poll_s 12:00 ?      00:00:00 sshd: garth@pts/0
0 S garth     49756 49755 3 80 0 - 29351 wait   12:00 pts/0    00:00:00 -bash
0 R garth     49960 49756 0 80 0 - 26557 -      12:01 pts/0    00:00:00 ps -elf
0 S garth     49961 49756 0 80 0 - 24365 pipe_w 12:01 pts/0    00:00:00 grep --color=auto garth

garth@brutha:~$ pstree 49755
sshd--bash--pstree
garth@brutha:~$ ps auxwf |grep garth
root      49740  0.0  0.0  92352  4068 ?        Ss   12:00   0:00 \_ sshd: garth [priv]
garth     49755  0.0  0.0  92352  1696 ?        S    12:00   0:00 \_ sshd: garth@pts/0
garth     49756  1.1  0.0 117404 10576 pts/0    Ss   12:00   0:00 \_ -bash
garth     50071  0.0  0.0 107152  2140 pts/0    R+   12:01   0:00 \_ ps auxwf
garth     50072  0.0  0.0  97460   920 pts/0    S+   12:01   0:00 \_ grep --color=auto garth

garth@brutha:~$
```

- Variables and the Shell
- Processes
- Working with Processes
- IO Streams
- Pipes and Redirections

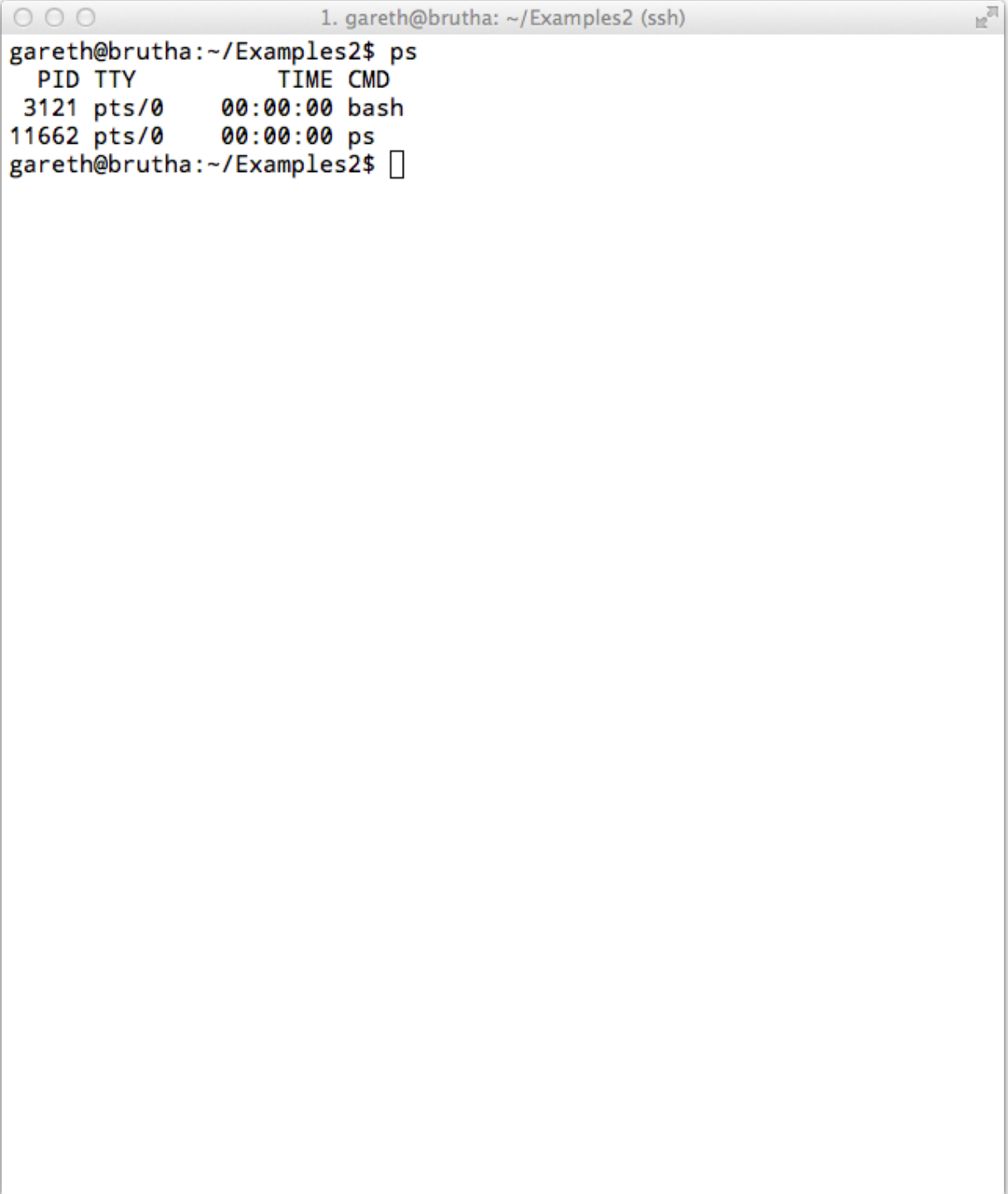
Common Tasks

- view the processes running on the system.
- start a new process in the background.
- move a process to the background.
- suspend a running process.
- kill a process.

[illegible]

Seeing processes

- You'll often need to monitor the status of processes running on your system.
- You can get a list of processes running on your system by using **ps**
- **ps** with no arguments will show you only the processes running in your shell
- **ps** shows the process id (**pid**), how long the command has been running and the command that is run.
- **ps -e** (or **ps -ax**) will show you all processes running on the system.
- other useful filters are:
 - **ps -elf**
 - **ps -auxwf**



```
1. gareth@brutha: ~/Examples2 (ssh)
gareth@brutha:~/Examples2$ ps
  PID TTY          TIME CMD
 3121 pts/0        00:00:00 bash
 11662 pts/0        00:00:00 ps
gareth@brutha:~/Examples2$
```

Other useful process tools

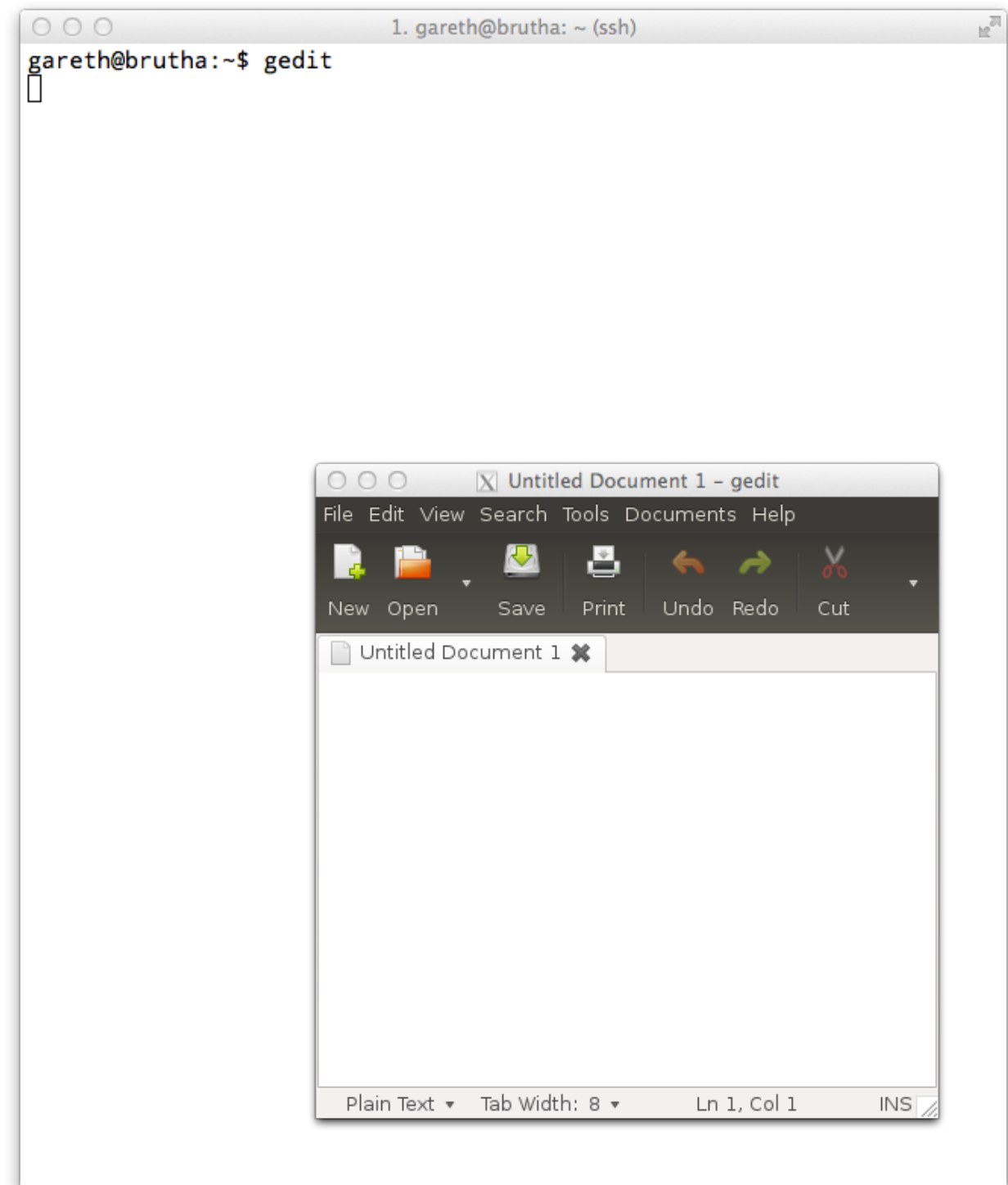
- If you want to see what processes are using particular resources you can use **top**.
- By default top sorts by **%CPU**.
- This can be changed by typing **F** and selecting a new sort field.
- To exit **top** type the **q** character.
- To see a hierarchical list of all the process we can use **pstree**.
- **pstree** shows the parent and children of each process in a graph. This lets us easily see the relationship between processes

```
1. gareth@brutha: ~/Examples2 (ssh)
top - 10:02:51 up 33 days, 13:29, 5 users, load average: 0.06, 0.07,
Tasks: 1249 total, 4 running, 1234 sleeping, 10 stopped, 1 zombie
Cpu(s): 0.1%us, 0.1%sy, 0.0%ni, 99.8%id, 0.0%wa, 0.0%hi, 0.0%si,
Mem: 264134840k total, 19521920k used, 244612920k free, 1240860k buf
Swap: 1000444k total, 0k used, 1000444k free, 10701224k cache

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 13758 gareth    20   0 18272 2256  972  R   1.0   0.0   0:00.13 top
 55841 dmclaugh  20   0 673m   70m  28m  S   1.0   0.0 49:17.05 plugin-co
15753 mdeans    20   0 3367m 199m 112m  S   1.0   0.1 37:36.07 chromium-
27165 mwood     20   0 662m   49m  25m  S   1.0   0.0 67:07.93 plugin-co
   942 root      20   0    0     0    0  S   0.0   0.0  0:22.73 kworker/2
 1576 mdeans    20   0 223m  122m 7164  S   0.0   0.0  9:17.35 x2goagent
 1615 avahi     20   0 34664 4260 1476  S   0.0   0.0 49:48.17 avahi-dae
 1762 root      20   0 15984  772  560  S   0.0   0.0 54:29.84 irqbalanc
 2286 lightdm   20   0 794m   23m  13m  S   0.0   0.0 131:20.14 unity-gre
50591 dmclaugh  20   0 1172m 311m  53m  S   0.0   0.1 30:16.79 firefox
61378 martynas  20   0 646m   42m  25m  S   0.0   0.0 39:17.52 plugin-co
    1 root      20   0 24672 2556 1344  S   0.0   0.0  0:15.47 init
    2 root      20   0    0     0    0  S   0.0   0.0  0:00.84 kthreadd
    3 root      20   0    0     0    0  S   0.0   0.0  0:08.19 ksoftirqd
    4 root      20   0    0     0    0  S   0.0   0.0  0:00.00 kworker/0
    5 root      0 -20    0     0    0  S   0.0   0.0  0:00.00 kworker/0
    8 root      20   0    0     0    0  S   0.0   0.0 72:31.30 rcu_sched
    9 root      20   0    0     0    0  S   0.0   0.0  6:38.74 rcuos/0
   10 root      20   0    0     0    0  S   0.0   0.0  5:49.49 rcuos/1
   11 root      20   0    0     0    0  S   0.0   0.0  2:01.68 rcuos/2
   12 root      20   0    0     0    0  S   0.0   0.0  0:37.26 rcuos/3
   13 root      20   0    0     0    0  S   0.0   0.0  1:50.76 rcuos/4
   14 root      20   0    0     0    0  S   0.0   0.0  5:04.32 rcuos/5
   15 root      20   0    0     0    0  S   0.0   0.0  1:53.96 rcuos/6
   16 root      20   0    0     0    0  S   0.0   0.0  5:32.29 rcuos/7
   17 root      20   0    0     0    0  S   0.0   0.0  0:54.64 rcuos/8
   18 root      20   0    0     0    0  S   0.0   0.0  0:21.13 rcuos/9
   19 root      20   0    0     0    0  S   0.0   0.0  0:38.66 rcuos/10
   20 root      20   0    0     0    0  S   0.0   0.0  1:18.35 rcuos/11
   21 root      20   0    0     0    0  S   0.0   0.0  0:43.65 rcuos/12
   22 root      20   0    0     0    0  S   0.0   0.0  0:11.34 rcuos/13
   23 root      20   0    0     0    0  S   0.0   0.0  0:44.64 rcuos/14
   24 root      20   0    0     0    0  S   0.0   0.0  0:29.06 rcuos/15
   25 root      20   0    0     0    0  S   0.0   0.0  0:16.16 rcuos/16
```

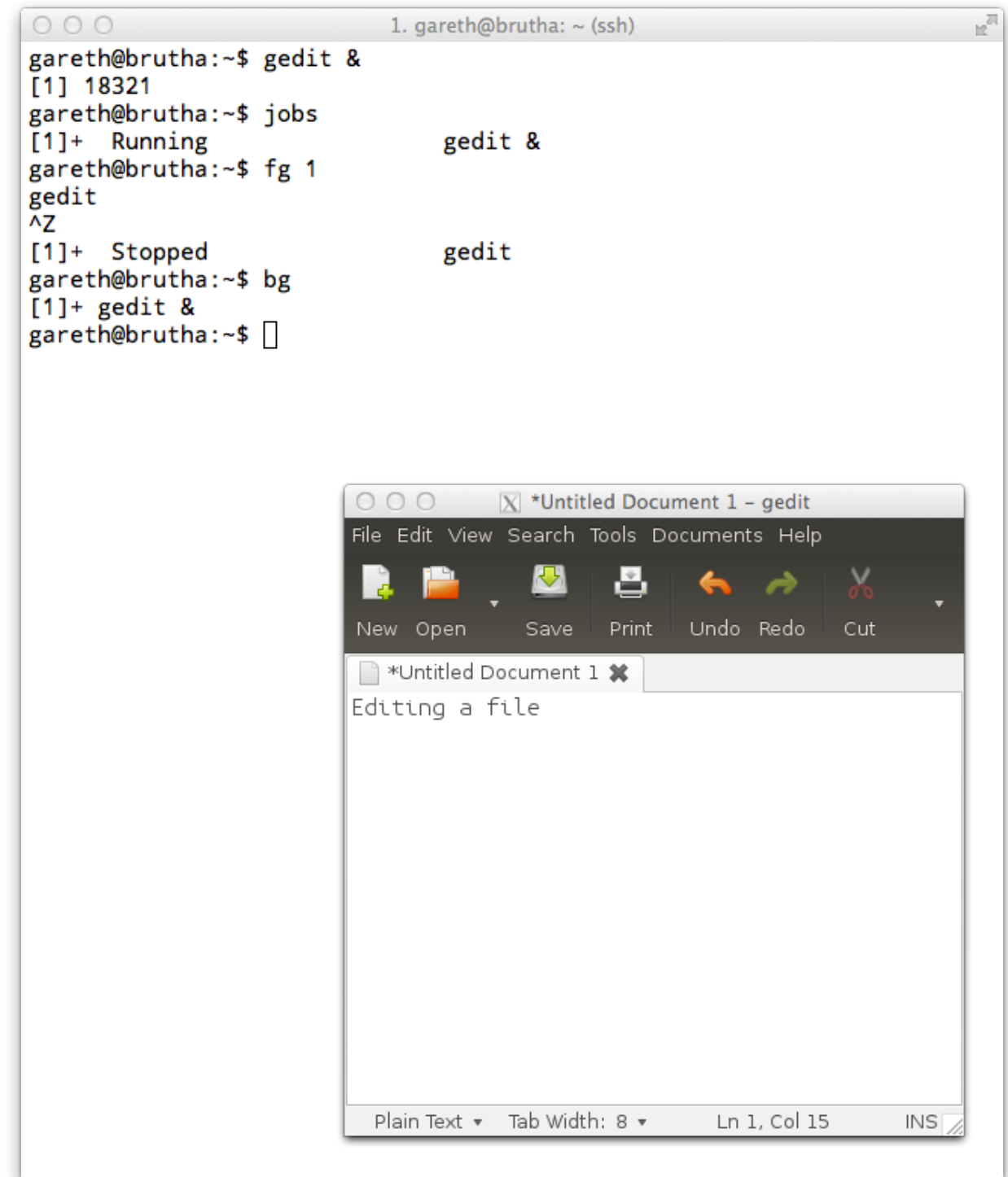
Foreground and Background

- Foreground jobs are processes that have interactive access to the command line.
- Background processes don't have access to the interactive command line but still run.
- Use '**&**' to start a process in the background and return to the command line for the next instruction.
- This can be used to start a text editor but allow you to continue to use the terminal.
- Use 'jobs' to check what processes are running in the current terminal.



Job Control

- **ctrl-c** - will kill a job that is currently running in the foreground.
- **ctrl-z** - will suspend a job that's currently running in the foreground.
- A suspended job retains its state but doesn't run or use CPU time.
- **fg** <job id> - will bring the specific job into the foreground.
- **bg** <job id> - will put a suspended job into the background



The image shows two overlapping windows from a Linux desktop environment. The top window is a terminal titled "1. gareth@brutha: ~ (ssh)". It contains the following text:

```
gareth@brutha:~$ gedit &
[1] 18321
gareth@brutha:~$ jobs
[1]+  Running                  gedit &
gareth@brutha:~$ fg 1
gedit
^Z
[1]+  Stopped                  gedit
gareth@brutha:~$ bg
[1]+  gedit &
gareth@brutha:~$
```

The bottom window is a gedit text editor titled "*Untitled Document 1 - gedit". It has a menu bar (File, Edit, View, Search, Tools, Documents, Help) and a toolbar with icons for New, Open, Save, Print, Undo, Redo, and Cut. The text area contains the text "Editing a file". The status bar at the bottom indicates "Plain Text", "Tab Width: 8", "Ln 1, Col 15", and "INS" mode.

Killing jobs

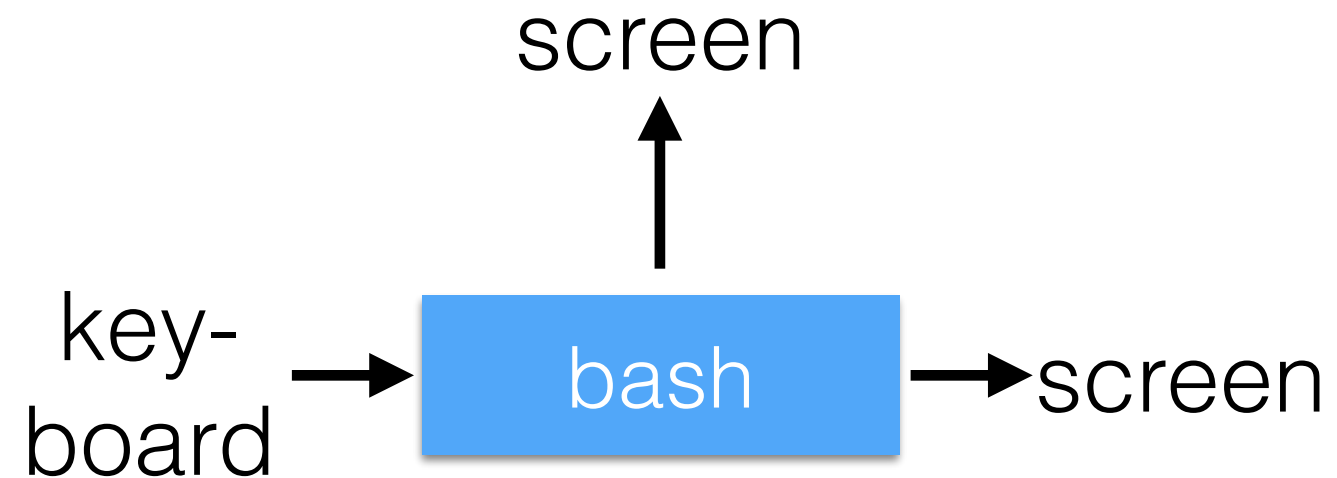
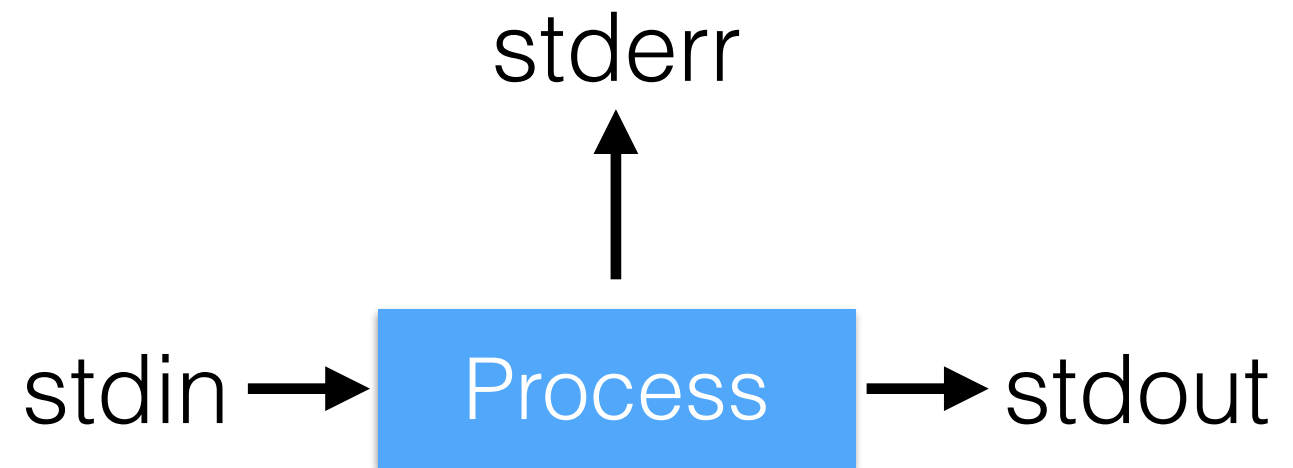
- Sometime we want to get rid of running processes.
- To remove a process from the system we use the **kill** command
 - **kill** <pid>
- You can find the pid either by running ps, or by using the pidof command
 - **pidof** <name>
- Sometimes we want to kill a process and all of it's children. to do this we use:
 - **kill -9** <pid>

```
1. gareth@brutha: ~/Examples2 (ssh)
gareth@brutha:~/Examples2$ ./infinite > /dev/null &
[1] 23300
gareth@brutha:~/Examples2$ jobs
[1]+  Running                  ./infinite > /dev/null &
gareth@brutha:~/Examples2$ ps
  PID TTY          TIME CMD
 17397 pts/0        00:00:00 bash
 17690 pts/0        00:00:00 dbus-launch
 23300 pts/0        00:00:08 infinite
 23337 pts/0        00:00:00 ps
gareth@brutha:~/Examples2$ pidof infinite
23300
gareth@brutha:~/Examples2$ kill 23300
gareth@brutha:~/Examples2$ ps
  PID TTY          TIME CMD
 17397 pts/0        00:00:00 bash
 17690 pts/0        00:00:00 dbus-launch
 23412 pts/0        00:00:00 ps
[1]+  Terminated              ./infinite > /dev/null
gareth@brutha:~/Examples2$
```

- Variables and the Shell
- Processes
- Working with Processes
- IO Streams
- Pipes and Redirections

Standard Streams

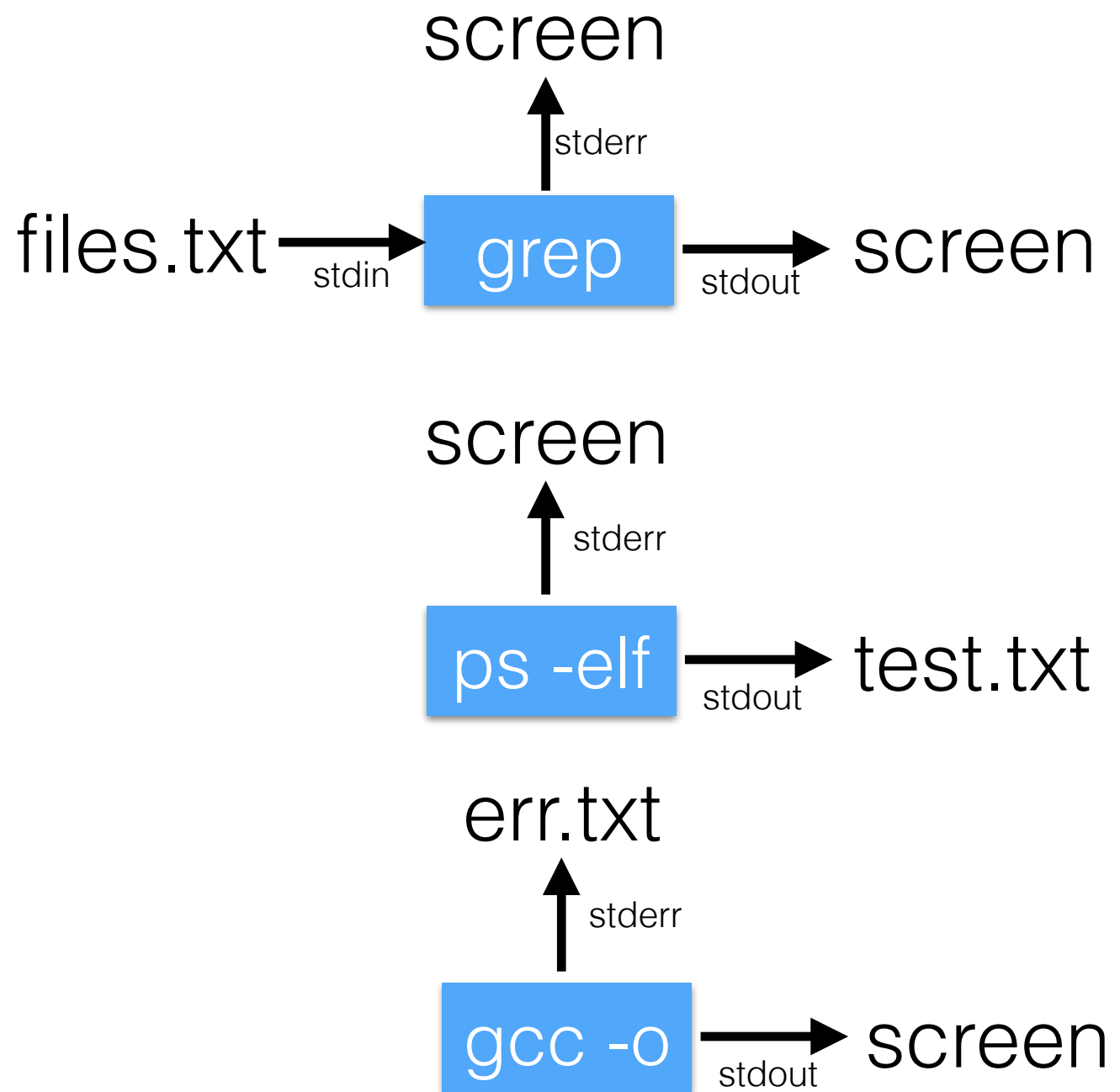
- Each process has three standard input/output streams.
- These come from the C standard and are built-in to every executable that has a “**main**” function.
- **stdin** - is the standard source of input to the program.
- **stdout** - is the standard output of the program, in the cases we have seen this is written to the screen/shell.
- **stderr** - is the standard error of the program, this allows errors to be handled separately from output.



- Variables and the Shell
- Processes
- Working with Processes
- IO Streams
- Pipes and Redirections

Redirection

- The standard streams (**stdin**, **stderr** and **stdout**) can be redirected to point to places.
- You can redirect stdin to put a list of typed commands into a program.
- You can redirect stdout to save the output of a program to a file for later processing.
- You can redirect stderr to save the error messages to look at later, or fix.



Redirection

- On the command line we redirect via the following commands:
- **<** - redirect stdin from a file or another stream.
- **1>** or **>** - redirect stdout to a file or another stream. If the file didn't exist it will be created, if it did exist it will be overwritten.
- **2>** - redirect stderr to a file or another stream
- **>&1** - redirect to stdout (i.e. **2>&1** to redirect stderr to stdout)
- **>&2** - redirect to stderr
- **>>** - append rather than overwrite.

```
1. gareth@brutha: ~/Examples2 (ssh)
gareth@brutha:~/Examples2$ grep for < loop2.c
    for (i=0; i < length; i++){
        for (i=1; i <= length; i++) {
gareth@brutha:~/Examples2$ ps > process.txt
gareth@brutha:~/Examples2$ cat process.txt
  PID TTY          TIME CMD
 17397 pts/0        00:00:00 bash
 17690 pts/0        00:00:00 dbus-launch
 43873 pts/0        00:00:00 ps
gareth@brutha:~/Examples2$ gcc
gcc: fatal error: no input files
compilation terminated.
gareth@brutha:~/Examples2$ gcc 2> errors.txt
gareth@brutha:~/Examples2$ cat errors.txt
gcc: fatal error: no input files
compilation terminated.
gareth@brutha:~/Examples2$
```

Pipes

- We often want to take the output of one command and use it as the input of another command.
- Using redirection we could do something like:
 - **ps -elf > temp.txt && grep n_tty < temp.txt**
- This is such a common pattern on UNIX systems that the OS provides a system for doing this called pipes.
- The above command could be re-written as:
 - **ps -elf | grep n_tty**
- Where the | character means take the stout of the first command and attach it to the second.
- This allows us to chain together multiple commands.

```
1. gareth@brutha: ~/Examples2 (ssh)
gareth@brutha:~/Examples2$ ps -elf > temp.txt && grep n_tty < temp.txt
4 S root      1705      1  0  80   0 - 25964 n_tty_  2014 tty4    00:0
0:00 /sbin/getty -8 38400 tty4
4 S root      1713      1  0  80   0 - 25964 n_tty_  2014 tty5    00:0
0:00 /sbin/getty -8 38400 tty5
4 S root      1732      1  0  80   0 - 25964 n_tty_  2014 tty2    00:0
0:00 /sbin/getty -8 38400 tty2
4 S root      1733      1  0  80   0 - 25964 n_tty_  2014 tty3    00:0
0:00 /sbin/getty -8 38400 tty3
4 S root      1736      1  0  80   0 - 25964 n_tty_  2014 tty6    00:0
0:00 /sbin/getty -8 38400 tty6
4 S root      2530      1  0  80   0 - 25964 n_tty_  2014 tty1    00:0
0:00 /sbin/getty -8 38400 tty1
0 S 2080089M  7540   7533  0  80   0 - 28189 n_tty_  Jan21 pts/5    00:0
0:00 bash
0 S mdeans    14281 14243  0  80   0 - 28236 n_tty_  Jan20 pts/13   00:0
0:00 bash
0 S mdeans    27672 14243  0  80   0 - 28239 n_tty_  Jan22 pts/12   00:0
0:00 bash
0 S mwood     35105 35082  0  80   0 - 28278 n_tty_  Jan20 pts/11   00:0
0:01 /bin/bash
0 S sams      45877 45866  0  80   0 - 25454 n_tty_  11:48 pts/15   00:0
0:00 pager -s
0 S 2083412S  47319 47312  0  80   0 - 28200 n_tty_  Jan23 pts/29   00:0
0:00 /bin/bash
0 S 20878010  58152 58145  0  80   0 - 29356 n_tty_  Jan23 pts/14   00:0
0:00 /bin/bash
gareth@brutha:~/Examples2$ ps -elf |grep n_tty |wc
      15      246     1366
gareth@brutha:~/Examples2$
```

- Variables and the Shell
- Processes
- Working with Processes
- IO Streams
- Pipes and Redirections