

CITS4407 Open Source Tools and Scripting

Version control, processes and pipelines

Unit coordinator: Arran Stewart

Overview

This week:

- files and filesystems – how does the Unix approach differ from, say, Windows?
- processes – what are they?
- what is version control, and why should we use it?

Files and filesystems

- Some things about a Unix system are not too dissimilar to Windows.
- For instance – users' data is stored in files, files are stored in directories, and both are stored on disks.
- But some other aspects are quite different.

“Everything is a file”

The Unix philosophy is to treat almost *everything* as a file.

- On Windows – if you want to see what tasks are running, there is a task manager program you can run which gives you this information.
- If you want to see what devices (like microphones or USB drives) are plugged in, there is a device manager program you can run.

“Everything is a file”

On Linux – all that information is visible by looking at the content of particular files.

(These are *virtual files*, and are said to live in a *virtual filesystem*. This just means they don't represent an actual file stored on disk, but rather present a “view” of some aspect of the operating system.)

- For convenience, Linux does also have graphical (and command-line) programs much like Windows's task manager and device manager;
- however, they're not *necessary* – you *could* get all the information you needed just by looking in particular files.

In a lab, try typing “`ls /proc`” sometime. The `/proc` directory holds details of all the programs that are currently running.

Processes

- To be more precise, what the `/proc` directory lists is details of *processes*.
- To a first approximation, you can think of a process as “a running program” . . .
- But processes aren’t necessarily “running”.
- They could be running, or they could be
 - waiting for data from a device like a disk drive
 - sleeping, because the process is waiting for an event like the user hitting a key on the keyboard
 - stopped (which you might think of as like “suspended”) – usually because the user deliberately stopped the process.

Processes

- We can also see what processes are running by using the command “ps”.

```
arran@barkley:cits4407-website$ ps
```

PID	TTY	TIME	CMD
2008	pts/24	00:00:00	ps
21347	pts/24	00:00:00	bash
31247	pts/24	00:00:08	evince

(Evince is a PDF viewer on Ubuntu Linux.)

Processes

By default, `ps` only list processes that are part of what is called “the current session” (programs launched from the terminal, or terminal window, you’re currently in).


`ps -A` lists *all* processes on the computer. Not just commands run by you, the user, but also what are called *daemons* – programs started by the operating system, and which run constantly “in the background”.

(Windows has an equivalent notion; on Windows, programs like this are called “services”.)

Processes and pipelines

One feature pioneered by Unix¹ is the idea of what are called *pipelines* between processes.

Often, we will want to perform multiple operations on a file – we might have student information stored in files, and might want to extract student names and marks; then sort the result of that, in descending order, by mark; then take just the top five students listed; and then send *that* list to a printer (or a program that, say, converts the result to PDF format).

¹Although the idea had appeared independently before. 

Processes and pipelines

We *could* do the extraction of names and marks, and store the result in a file; and then sort that file, and put the sorted result in a second file; and then run a command which just gives us the first five students.

But Bash allows us to avoid explicitly creating all those intermediate files, and instead “chain” commands together – *piping* the result of one command to another command.

Processes and pipelines

Piping in bash is done using what is often called (by programmers) the “pipe” character, “|”.

(Usually found just above the “enter” key, on English-language keyboards.)

Processes and pipelines

The sequence of commands to extract names and marks, sort, and get the first five lines could be done put in a “pipeline” like this:

```
$ cut -f 1,4 marks.txt | sort -n --key 2 | head -n 5
```

We'll see how to “compose” pipelines in the lab/workshops.

Version Control, Git, and GitHub

Version control

At its simplest, a *version control system* (VCS) lets you track how a set of files change over time, and lets you “roll back” to previous points in their history.

More complex things you might do are

- “roll back” to some previous point in time, and then “branch off”, making different changes to the ones you made previously
- delete “branches” of history you no longer need
- *merge* branches of history – combining the changes made in two separate branches.

Version control

Often, people working with files end up creating “informal” version control systems themselves – keeping multiple versions of a file like

- AlgorithmsAssignment1Draft.doc
- AlgorithmsAssignment1FinalVersion.doc
- AlgorithmsAssignment1FinalFinalVersion.doc
- AlgorithmsAssignment1FinalFinalVersion (02).doc

so that they can get back to a previous version if they need to.

Version control

But this kind of informal “version control system” relies on us remembering what each file name meant, and how it related to other files – and it becomes harder to manage if multiple people are trying to make many changes to the files at once.

But that sort of situation is exactly the case when writing software (or managing business data).

Version control

So we use *software* version control systems, which track

- when a file was changed, and how
- who changed it
- *why* they changed it (by allowing users to leave *comments* about the changes they have made)
- how the change relates to other changes – what changes came before and after it.

Advantages of version control

If we ever need to

- take all our files back to the state they were in on 1 March, 2020
- look at conflicting changes two users have made to a file, and decide which should take precedence
- work out exactly which change introduced a bug which crashed our system last week

then version control systems let us do this.

Git

We will suggest using a version control system call *Git*, created by Linus Torvalds (the creator of Linux) for tracking changes made to the Linux kernel, in 2005.

Terminology

Some terminology –

- a *repository* (or “repo”) is a set of files you wish to track
- a repository may be stored on a remote *server* – some computer that stores the files
- a *commit* (or “revision”) is a state of the files at a particular point in their “history”
- *cloning* a repository means to make a copy of it (including all the history details etc it contains) – typically the new repository will keep a record of where it was cloned from

Very basic Git use

An example of use:

```
$ cd MyProjects  
$ mkdir assignment1  
$ cd assignment1  
$ git init
```

```
git init
```

git init

Here we've create a directory to store files in, and *initialized* it as a Git repository.

That creates a hidden directory call “.git” within our assignment1 directory, and files in this hidden directory track all the information about what changes were made when.

Adding files

Suppose we now create a file in our `assignment1` directory (perhaps using a text editor) called “`fabulous-program.sh`”, and decide we now wish to keep this file under version control.

```
$ git add fabulous-program.sh  
$ git commit -m "initial version of the fabulous program"
```

Every time we make a change to the file which we wish to record, we repeat these commands.

Other Git operations

“init”, “add” and “commit” are examples of what are sometimes called *subcommands* – key words that specify a particular way you want to use a command, and which have *arguments* (the words that appear afterwards) of their own that control their behaviour.

In lab/workshops, we will see how to perform more complex operations – but if you know the init, add and commit commands, you can be assured that you can retrieve previous versions of your files.

GitHub

- Don't confuse the “git” command with **GitHub**
- “git” is an open source program; GitHub is a web-based hosting service for git repositories, currently owned by Microsoft
- GitHub is extremely popular, but there are many other competing hosting services:
 - BitBucket (<https://bitbucket.org/>)
 - GitLab (<https://gitlab.com>)
 - SourceHut (<https://sourcehut.org>)

Hosting services

- Also note that nothing about `git` *requires* that you use a hosting service – they are simply convenient ways of sharing a repository with other people.
- Some software projects host their own repositories (“hosting” a repository just means having a website through which the repository is accessible)
- Some projects and organisations don’t use Git at all – there are many other version control systems available (though Git is certainly one of the most popular)