LINUX VOICE MASTERCLASS

Essential Linux tools explained – this month, the Git version control system and its cloud-based cousin GitHub

GIT: VERSION CONTROL

Saved a file and changed your mind? Fear not, Git has your back...

JOHN LANE

Git is something that, as a Linux user, you'll come into contact with sooner or later. If you're compiling a package from its source code (perhaps after reading this issue's compiling tutorial on page 86), for instance, you may well have to use a command like this:

```
$ git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

This gets you the source code of the Linux kernel. And that is where the Git story begins – Linus Torvalds created Git to hold the Linux kernel sources. But our one-line example doesn't only get the latest version: you get the entire history too. Git is a version control system (VCS) that tracks changes to files and can reproduce a project's state as it was at any point in time. You can use it for your own projects too, giving you the ability to time-travel into its history or revert changes that you later wish you hadn't made.

Git stores files in a repository. This is the working copy of a project (its directory hierarchy of files) plus a hidden directory where Git efficiently stores the project's history. In contrast to earlier version control systems, Git has no central server. Instead, many people can clone a repository, each getting their own copy of everything. They each work in their own clone and can sync theirs with others. For this reason, Git is known as a distributed VCS.

LV PRO TIP

Git commands are well documented: use **man** or give the **--help** option to any Git command. List all commands with **git --help**.

Cloning is one way to get a repository. The other way is to start a new one. Let's say you have a project that you're working on and you have all its files in a **myproject** directory. Git calls this your working copy, because those are the files you work on as you develop your project. To track your project with Git, you first need to initialise your new repository in its root directory:

```
$ cd ~/myproject
$ git init
Initialized empty Git repository in /home/john/myproject
```

Git works by taking snapshots, called commits, of the files in your working copy. You control what should be included in each commit. Git maintains an index of the files in your working copy that you wish to commit and you add files to this index. Adding files to the index is called staging.

After initialising a new repository, it's customary to commit all files so that they are tracked:

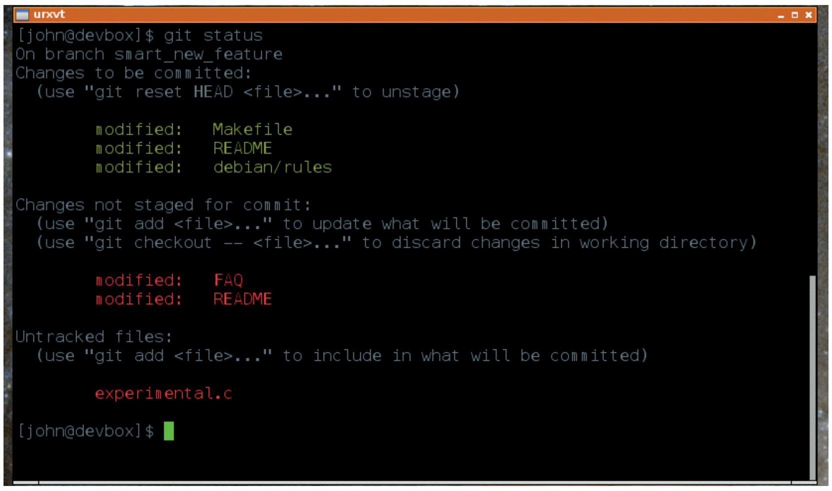
```
$ git add .
$ git commit -m "initial commit"
```

The first command adds everything within and beneath the current directory to the index, and the second command commits them. Each commit requires a descriptive commit message. The **-m** argument enables you to give this on the command line, otherwise Git will open your default editor for you to provide one.

Each commit is a new snapshot of your project that's made from the snapshot made for the previous commit plus whatever is in the index when the commit is done. Therefore each commit records a snapshot of your project as it was at that point in time (don't worry – it does this very efficiently). Committing empties the index, so any future changes need to be added to the index again. This action is called staging, and it enables you to control, per-commit, which changes are included. Staging is also done with **git add**. You can specify a single file or use wildcards

```
$ git add hello.c
$ git add *.c
```

When you stage a file, Git takes a snapshot of it, and it is this snapshot that you later commit. If you change



```
[john@devbox]$ git status
On branch smart_new_feature
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   Makefile
        modified:   README
        modified:   debian/rules

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   FAQ
        modified:   README

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        experimental.c

[john@devbox]$
```

The **git status** command shows what is staged for the next commit. The **README** file has both staged and unstaged changes, and **experimental.c** is untracked.

a staged file before committing it and want those changes included, you must stage it again (using **git add** again).

One extremely useful command is **git status**. It shows at a glance the staged files that will be included in the next commit and any tracked files with unstaged changes that won't be included. It also shows you any files in the working copy that are untracked. Staged files that are changed before committing will show twice: as a file to be committed and also as one with unstaged changes.

Each commit is given a unique reference: a SHA1 hash of its contents, which you can see with **git log**. Any commit can be recovered using its SHA1 (actually, you don't need to use all of it – just enough characters to make it unique). These can still be difficult to remember though, so Git has another kind of reference – the branch.

Branching out

A branch is the list of commits from the latest right back to the initial commit made when the repository was created. New commits are added to the top, or HEAD, of the current branch.

The output of **git status** shows the name of the current branch, and new repositories have a single branch called **master**. You would typically create a new branch for a specific piece of work and merge it into the master branch once it's complete. This avoids polluting your master with incomplete changes and unwanted changes can be easily abandoned.

Branches deviate from the commit where they are made in a tree-like fashion, and commits on one branch do not impact others unless merged. You can switch between branches with **git checkout** – useful if you're working on multiple unrelated changes at the same time, or you can use the **-b** option to check out a new branch from the head of the current branch.

```
$ git checkout -b mybranch
Switched to a new branch 'mybranch'
```

You can list all branches and see the current branch, marked with an asterisk:

```
$ git branch
  master
* mybranch
  myotherbranch
```

and check out whichever one you want

```
$ git checkout master
Switched to branch 'master'
```

To merge another branch into your current one:

```
$ git merge mybranch
```

and to delete a branch after merging (or if it's unwanted):

```
$ git branch -d mybranch
```

When you check out, Git replaces what's in your working copy with what was there at the time of the commit you check out.

As you commit, branch and merge, it can be useful to visualise your project's commit tree, and there are a number of ways to do this. The simplest of these runs on the command line and uses ASCII-art to represent commits.

```
$ git log --graph
```

But this can be improved upon with Git's instant web server. Start it in your project's root directory (it requires the lighttpd web server, or you can use **--httpd** to specify another server):

```
$ git instaweb
```

It should open a browser window for you, or you can navigate to **http://localhost:1234**. Better still is the repository browser: just enter **gitk** to open it.

You are not alone

So far you've been working in your own repository, but you may want to share work if you're collaborating with others. Git enables you to fetch from and push to other repositories. First, you need to tell it where these remote repositories are:

```
$ git remote add name URL
```

The **name** can be anything that describes the remote – perhaps a colleague's name. The URL describes how to access it and Git supports various protocols including HTTP, HTTPS, its own "git" protocol, direct file access to the local filesystem and SSH. The latter offers authentication and is usually required when pushing changes to a remote.

If your repository is a clone of another it will already have a remote (referred to as its **origin**) pointing to where it was cloned from. Pushing changes to it would be done like this:

```
$ git push origin master
```

You can fetch from a remote and this gives you remote branches that you can list. Remote branches are distinct from your local branches.

```
$ git fetch myremote
$ git branch -r
```

After fetching a remote, you can merge a remote branch into your current one:

```
$ git merge myremote/branch
```

This is a common task and Git provides a shortcut for it. Another way to achieve a fetch and merge is to pull:

```
$ git pull myremote branch
```

First among equals

We said that all repository clones are equal, but you can still treat one as the master copy and have people clone that and push their changes to it. It's common to locate such a repo on a server that is backed up, perhaps in the cloud to make it accessible. There are many options here -- consider Gitolite, Gitorious or CGit on your own server or use a cloud service like Bitbucket, Sourceforge or the one we'll explore next: GitHub.

The Pro Git book is an excellent resource, and you can read it online for free at **http://git-scm.com/book**.

"If you're working with others, Git enables you to fetch from and push to other repositories."

LV PRO TIP

List any files or directories that you don't want tracked in a file called **.gitignore** and Git will ignore them.

LV PRO TIP

Git branches are unlike other VCS – nothing needs to be copied and this makes them fast and lightweight.

GITHUB: GIT IN THE CLOUDS

Publicise your project, take a free backup and other nice things. All with GitHub…

GitHub is probably the world's largest code host. It enables you to host publicly-accessible Git repositories as well as, for a monthly fee, private ones. Many open-source projects use GitHub as their online presence.

Read access to public repositories is free and does not require registration, and many open-source projects offer the GitHub URL of their repository, which can be cloned without ever using the GitHub website. But if you do this you'd be missing out, as the website is a great way to browse repositories – it has an intuitive interface and nice features like syntax highlighting of source code.

You need a GitHub account to host your own work. If you don't have one then head over to **github.com**, choose a username and sign up. Then you can create a repository. You then need to choose a name specific to your account, and can enter a longer description if you want to. Next, you choose whether the new repository is private, which incurs fees, or public.

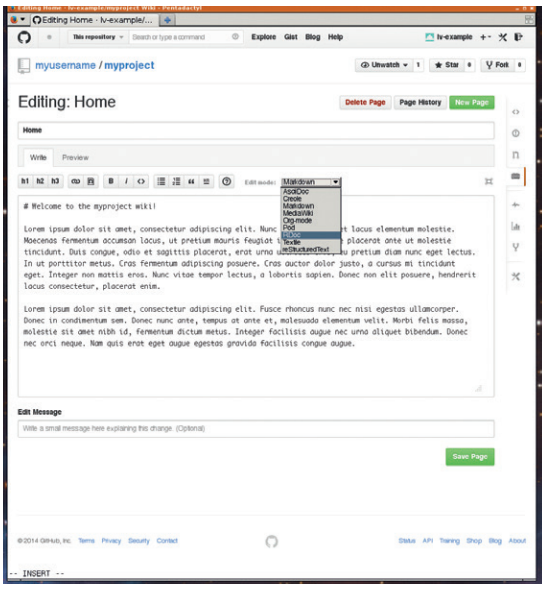
There are a few other options that you can select. You can initialise the new repository with **README** and **.gitignore** files, and you can also choose an open-source licence file. If you have already initialised a repository locally (with **git init**) then you should leave the GitHub one empty, because you will set it up as a remote and push to it.

GitHub then presents you with the exact commands needed to create a new repository or configure an existing one so that you can push it up to GitHub. Continuing our conceptual **myproject** example, we would add GitHub as a remote:

```
cd ~/myproject
$ git remote add origin https://github.com/myusername/
myproject.git
```

LV PRO TIP

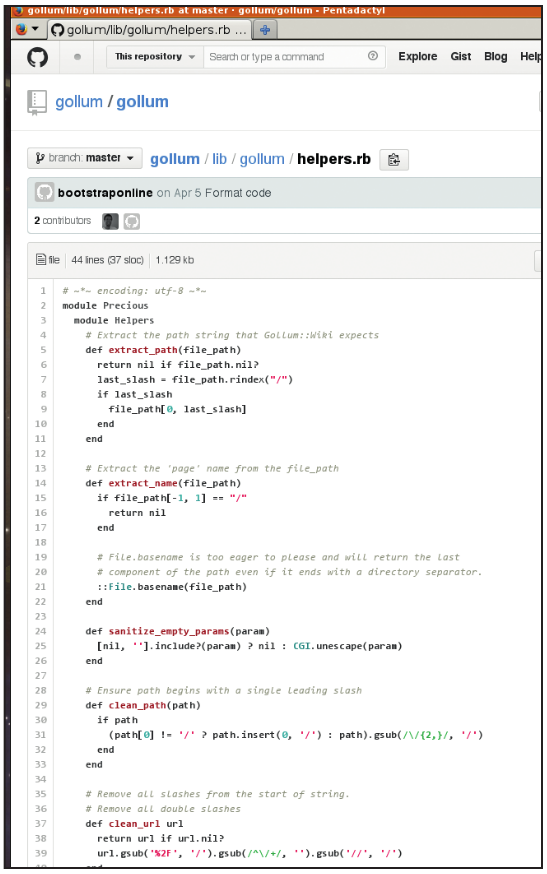
You can use GitHub-flavoured Markdown (**bit.ly/13jiaLV**) to format your comments on issues and pull requests.


```
# Welcome to the myproject wiki!

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc [illegible]t lacus elementum molestie.
Maecenas fermentum accumsan lacus, ut pretium mauris feugiat i[illegible] placerat ante ut molestie
tincidunt. Duis congue, odio et sagittis placerat, erat urna u[illegible]u pretium diam nunc eget lectus.
In ut porttitor metus. Cras fermentum adipiscing posuere. Cras auctor dolor justo, a cursus mi tincidunt
eget. Integer non mattis eros. Nunc vitae tempor lectus, a lobortis sapien. Donec non elit posuere, hendrerit
lacus consectetur, placerat enim.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce rhoncus nunc nec nisi egestas ullamcorper.
Donec in condimentum sem. Donec nunc ante, tempus at ante et, malesuada elementum velit. Morbi felis massa,
molestie sit amet nibh id, fermentum dictum metus. Integer facilisis augue nec urna aliquet bibendum. Donec
nec orci neque. Nam quis erat eget augue egestas gravida facilisis congue augue.
```

GitHub's Wiki editor supports many kinds of markup


```
# ~*~ encoding: utf-8 ~*~
module Precious
  module Helpers
    # Extract the path string that Gollum::Wiki expects
    def extract_path(file_path)
      return nil if file_path.nil?
      last_slash = file_path.rindex("/")
      if last_slash
        file_path[0, last_slash]
      end
    end

    # Extract the 'page' name from the file_path
    def extract_name(file_path)
      if file_path[-1, 1] == "/"
        return nil
      end

      # File.basename is too eager to please and will return the last
      # component of the path even if it ends with a directory separator.
      ::File.basename(file_path)
    end

    def sanitize_empty_params(param)
      [nil, ''].include?(param) ? nil : CGI.unescape(param)
    end

    # Ensure path begins with a single leading slash
    def clean_path(path)
      if path
        (path[0] != '/' ? path.insert(0, '/') : path).gsub(/\/{2,}/, '/')
      end
    end

    # Remove all slashes from the start of string.
    # Remove all double slashes
    def clean_url url
      return url if url.nil?
      url.gsub('%2F', '/').gsub(/^\/+/, '').gsub('//', '/')
```
gollum/lib/gollum/helpers.rb at master · gollum/gollum - Pentadactyl
This repository
Search or type a command
Explore
Gist
Blog
Help
gollum / gollum
branch: master
gollum / lib / gollum / helpers.rb
bootstraponline on Apr 5 Format code
2 contributors
file 44 lines (37 sloc) 1.129 kb

Browse repositories on GitHub with syntax highlighting

```
$ git push -u origin master
```

The **-u** argument tells Git to remember that **origin/master** is the remote, or upstream branch, which the local branch pushes to. It allows future pushes to omit the remote details:

```
$ git push
```

You'll be prompted for your GitHub username and password each time you push, but you can avoid this by using SSH instead of HTTP. You'll need to add your SSH public key to your GitHub account – click the tools icon on the top right-hand corner of the page, go to SSH Keys and paste in the text of your public key. You can then change the remote's URL from HTTP to SSH like this:

```
$ git remote set-url origin git@github.com:myusername/
myproject.git
```

Fork and pull

You can quite happily get along using GitHub as a remote repository for your work, allowing you to share it with the world or even just as a backup (bear in mind that anyone can view fee-free repositories). However, there is much more on offer. If you want to contribute to a repository that is not your own, you can fork it. This gives you your own copy of that repository that you can then clone to your local machine and make those great changes that you have in mind without disturbing anyone else. To fork another repository, first locate it using the search tool, then just click the fork icon.

When you want to share those changes, you issue a pull request. This sends a formatted patch to the owner of the repository that you forked. It is then up to them to review and accept your patch. You first commit your local changes and push them to your local fork on GitHub. Then, on the GitHub screen for that repository, click on Pull Request to generate a patch and send the pull request. Anyone can leave comments on a pull request and this offers a good way to discuss it prior to it being accepted.

Wiki pages

GitHub includes a wiki system called Gollum, which provides wiki pages on every repository. You decide whether to have one and you can apply some limited editing controls. There's a comprehensive editor right there on the site or you can clone your wiki to your own machine and work locally:

```
$ git clone git@github.com:myusername/myproject.wiki
```

Gollum supports quite a few markup syntaxes, so if Markdown isn't your thing, you could try one of several others on offer including MediaWiki, RDoc and ReStructuredText.

The wiki is one way to provide content for your repository, but there is another called GitHub Pages, which enables you to add a static website to your repository. You can include content by adding a branch named **gh-pages** and committing content to it. With your repository in a clean state (**git status** reports "nothing to commit, working directory clean"):

```
$ git checkout --orphan gh-pages
$ git rm -rf .
```

This checks out a new empty branch that is unrelated to your existing commits, which makes sense because it will contain unrelated content. You need to clean out the working copy yourself, however, before starting to write your site's content – perhaps beginning with a new **index.html** file. To upload it, commit and push to a new remote branch:

```
$ git commit -m "first commit of site pages"
$ git push -u origin gh-pages
```

When the **gh-pages** branch is in place, GitHub will publish it automatically and it should be accessible within a few minutes. The repository's settings will show the site's address, which will be something like **http://myusername.github.io/myproject**. You can even use your own domain by writing a **CNAME** file in the root directory of your **gh-pages** branch. Put your domain in this file:

```
myproject.mydomain.co.uk
```

You will also need to configure your domains DNS, and you can read more about this at **http://bit.ly/ghdomain**.

Each repo also has an issues page, where anyone can log issues against it. Similarly to pull requests, anyone can comment on issues. Posts can be given labels like bug, enhancement or question. You can, however, disable a repository's issues tracker on its settings page. Some projects do this because they prefer to receive pull requests.

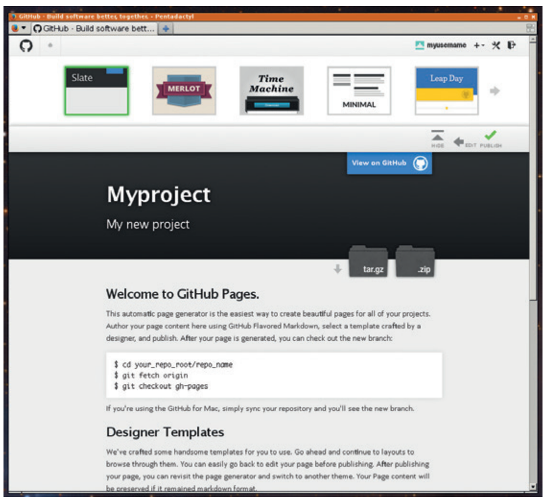
In addition to repository sites (Project Pages as GitHub calls them), you can have User Pages too. They work in exactly the same way, but you need to create a repository called **myusername.github.io** and store your content in its master branch. User pages are served at **http://mysername.github.io**. You should note that GitHub Pages are served over HTTP, so are not suited to sensitive information.

Get the Gist

And, if all that weren't enough, GitHub also provides a pastebin service, called Gist, which is a way to quickly share snippets of code and other pastes. Once again, you can use Markdown and it also has syntax highlighting. Gists are themselves Git repositories and can therefore be cloned (the required URL is presented on the gist's page). As a GitHub user, you automatically have a gist site at **gist.github.com/myusername**, or you can click the link at the top of your GitHub page.

If you have your own projects, no matter how small, GitHub is a handy addition to your toolkit, especially if you're already a Git user. If you aren't, GitHub makes it easier to become one. LV

"If you want to contribute to a repository that is not your own, you can fork it."



Myproject

My new project

Welcome to GitHub Pages.

This automatic page generator is the easiest way to create beautiful pages for all of your projects. Author your page content here using GitHub Flavored Markdown, select a template crafted by a designer, and publish. After your page is generated, you can check out the new branch:

```
$ cd your_repo_root/repo_name
$ git fetch origin
$ git checkout gh-pages
```

If you're using the GitHub for Mac, simply sync your repository and you'll see the new branch.

Designer Templates

We've crafted some handsome templates for you to use. Go ahead and continue to layouts to browse through them. You can easily go back to edit your page before publishing. After publishing your page, you can revisit the page generator and switch to another theme. Your Page content will be preserved if it remained markdown format.

If you don't want to write your own pages by hand, you can use GitHub's automatic page generator. This offers a number of themes and an editor that can get you published quickly.

LV PRO TIP

You can clone GitHub's wiki system and use it yourself: **https://github.com/gollum**.

Also consider

There are alternatives to GitHub. You might want to check out Gitorious or GitLab; both offer hosted services similar to GitHub but release open-source community editions that offer the option of installing on your own server.

John Lane is a technology consultant with a penchant for Linux. He helps new businesses start-ups make the most of open source software.