An introduction to version control with Git

Sam Fearn
(s.m.fearn@durham.ac.uk)

November 5th, 2019

Outline

1. Introduction

2. How to actually use git

3. Useful Resources

What is Version Control?

- A Version Control System (VCS, sometimes also SCMS) is a tool for managing a changing collection of files, allowing you to get back to a particular version at any time.

What is Version Control?

- A Version Control System (VCS, sometimes also SCMS) is a tool for managing a changing collection of files, allowing you to get back to a particular version at any time.
- E.g. myfile041119.txt, myfile051119.txt, ...

What is Version Control?

- A Version Control System (VCS, sometimes also SCMS) is a tool for managing a changing collection of files, allowing you to get back to a particular version at any time.
- E.g. myfile041119.txt, myfile051119.txt, ...
- The method above has many problems, including duplication, naming errors, easy to accidentally overwrite the old file, ...

What is Version Control?

- A Version Control System (VCS, sometimes also SCMS) is a tool for managing a changing collection of files, allowing you to get back to a particular version at any time.
- E.g. myfile041119.txt, myfile051119.txt, ...
- The method above has many problems, including duplication, naming errors, easy to accidentally overwrite the old file, ...
- Also, if you're collaborating on a project, this can get very tricky to manage. Which version did I last send? Have I included the latest changes from my collaborator? What do I do if they make changes to a section I've also changed since we last compared?

What is Version Control?

- A Version Control System (VCS, sometimes also SCMS) is a tool for managing a changing collection of files, allowing you to get back to a particular version at any time.
- E.g. myfile041119.txt, myfile051119.txt, ...
- The method above has many problems, including duplication, naming errors, easy to accidentally overwrite the old file, ...
- Also, if you're collaborating on a project, this can get very tricky to manage. Which version did I last send? Have I included the latest changes from my collaborator? What do I do if they make changes to a section I've also changed since we last compared?

Different VCS

- Many different VCS have been created to address the challenges of managing different versions of files.

Different VCS

- Many different VCS have been created to address the challenges of managing different versions of files.
- Most of the popular systems in use today are either **Centralised Version Control Systems**, or **Distributed Version Control Systems**.

Different VCS

- Many different VCS have been created to address the challenges of managing different versions of files.
- Most of the popular systems in use today are either **Centralised Version Control Systems**, or **Distributed Version Control Systems**.
- With a CVCS, there is a single server which manages the versioned files – the main downside is that this type of system has a single point of failure.

Different VCS

- Many different VCS have been created to address the challenges of managing different versions of files.
- Most of the popular systems in use today are either **Centralised Version Control Systems**, or **Distributed Version Control Systems**.
- With a CVCS, there is a single server which manages the versioned files – the main downside is that this type of system has a single point of failure.
- Examples of CVCS you may have heard of include SVN and Subversion.

Different VCS

- Many different VCS have been created to address the challenges of managing different versions of files.
- Most of the popular systems in use today are either **Centralised Version Control Systems**, or **Distributed Version Control Systems**.
- With a CVCS, there is a single server which manages the versioned files – the main downside is that this type of system has a single point of failure.
- Examples of CVCS you may have heard of include SVN and Subversion.
- In a DVCS, the entire history of each project is mirrored across multiple clients – this protects against the single point of failure.

Different VCS

- Many different VCS have been created to address the challenges of managing different versions of files.
- Most of the popular systems in use today are either **Centralised Version Control Systems**, or **Distributed Version Control Systems**.
- With a CVCS, there is a single server which manages the versioned files – the main downside is that this type of system has a single point of failure.
- Examples of CVCS you may have heard of include SVN and Subversion.
- In a DVCS, the entire history of each project is mirrored across multiple clients – this protects against the single point of failure.
- Common examples of DVCS include Git and Mercurial.

Different VCS

- Many different VCS have been created to address the challenges of managing different versions of files.
- Most of the popular systems in use today are either **Centralised Version Control Systems**, or **Distributed Version Control Systems**.
- With a CVCS, there is a single server which manages the versioned files – the main downside is that this type of system has a single point of failure.
- Examples of CVCS you may have heard of include SVN and Subversion.
- In a DVCS, the entire history of each project is mirrored across multiple clients – this protects against the single point of failure.
- Common examples of DVCS include Git and Mercurial.
- Of all the systems mentioned above, Git is probably the most common and widely used.

Different VCS

- Many different VCS have been created to address the challenges of managing different versions of files.
- Most of the popular systems in use today are either **Centralised Version Control Systems**, or **Distributed Version Control Systems**.
- With a CVCS, there is a single server which manages the versioned files – the main downside is that this type of system has a single point of failure.
- Examples of CVCS you may have heard of include SVN and Subversion.
- In a DVCS, the entire history of each project is mirrored across multiple clients – this protects against the single point of failure.
- Common examples of DVCS include Git and Mercurial.
- Of all the systems mentioned above, Git is probably the most common and widely used.

Git and Github

- Github is a free git hosting service, allowing you to share repositories with other people (or keep them private) – alternatives do exist (Bitbucket, Kiln?).

Git and Github

- Github is a free git hosting service, allowing you to share repositories with other people (or keep them private) – alternatives do exist (Bitbucket, Kiln?).
- Git can be used without Github. Your repository is always stored locally, and you can share this however you wish – ssh, ftp, (email, memory stick,...!) etc.

Using Git at the command line

Creating a new repository

- `git init`

Using Git at the command line

Creating a new repository

- `git init`

Cloning an existing repository

- `git clone username@host:/path/to/repository`

Using Git at the command line

Creating a new repository

- `git init`

Cloning an existing repository

- `git clone username@host:/path/to/repository`

Check the **status** of a repository

- `git status`

Using Git at the command line

Creating a new repository

- `git init`

Cloning an existing repository

- `git clone username@host:/path/to/repository`

Check the **status** of a repository

- `git status`

Add files to the index

- `git add filename`

Using Git at the command line

Creating a new repository

- `git init`

Cloning an existing repository

- `git clone username@host:/path/to/repository`

Check the **status** of a repository

- `git status`

Add files to the index

- `git add filename`
- Or `git add *`

Using Git at the command line

Creating a new repository

- `git init`

Cloning an existing repository

- `git clone username@host:/path/to/repository`

Check the **status** of a repository

- `git status`

Add files to the index

- `git add filename`
- Or `git add *`
- Or `git add -u`

Using Git at the command line

Creating a new repository

- `git init`

Cloning an existing repository

- `git clone username@host:/path/to/repository`

Check the **status** of a repository

- `git status`

Add files to the index

- `git add filename`
- Or `git add *`
- Or `git add -u`

Commit the changes

- `git commit -m "Helpful message"`

Using Git at the command line

Creating a new repository

- `git init`

Cloning an existing repository

- `git clone username@host:/path/to/repository`

Check the **status** of a repository

- `git status`

Add files to the index

- `git add filename`
- Or `git add *`
- Or `git add -u`

Commit the changes

- `git commit -m "Helpful message"`

Add a remote server (if neccessary)

- `git remote add origin <server>`

Using Git at the command line

Creating a new repository

- `git init`

Cloning an existing repository

- `git clone username@host:/path/to/repository`

Check the **status** of a repository

- `git status`

Add files to the index

- `git add filename`
- Or `git add *`
- Or `git add -u`

Commit the changes

- `git commit -m "Helpful message"`

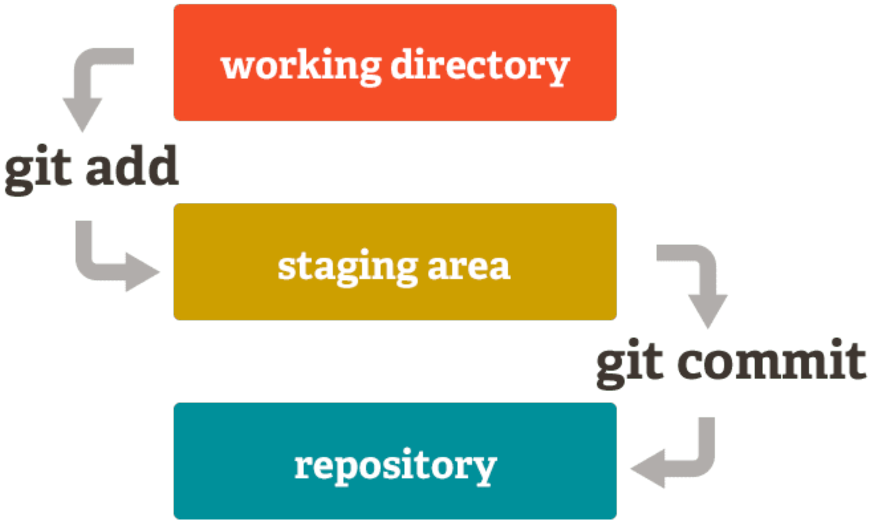
Add a remote server (if neccessary)

- `git remote add origin <server>`

Push the recent commits

- `git push origin master`

What's going on?



Updating to new changes

To update to the most recent commit

- `git pull`

Updating to new changes

To update to the most recent commit

- `git pull`

This syncs the local repository with the remote one, automatically merging changes

Updating to new changes

To update to the most recent commit

- `git pull`

This syncs the local repository with the remote one, automatically merging changes What if we want to work on different things at the same time?

Updating to new changes

To update to the most recent commit

- `git pull`

This syncs the local repository with the remote one, automatically merging changes What if we want to work on different things at the same time?

Create a new **branch**

- `git checkout -b <branch>`

Updating to new changes

To update to the most recent commit

- `git pull`

This syncs the local repository with the remote one, automatically merging changes What if we want to work on different things at the same time?

Create a new **branch**

- `git checkout -b <branch>`

After making some changes, **push** branch

- `git push origin <branch>`

Updating to new changes

To update to the most recent commit

- `git pull`

This syncs the local repository with the remote one, automatically merging changes What if we want to work on different things at the same time?

Create a new **branch**

- `git checkout -b <branch>`

After making some changes, **push** branch

- `git push origin <branch>`

Switch between branches

- `git checkout <branch>`

- `git merge <branch>`

Updating to new changes

To update to the most recent commit

- `git pull`

This syncs the local repository with the remote one, automatically merging changes What if we want to work on different things at the same time?

Create a new **branch**

- `git checkout -b <branch>`

After making some changes, **push** branch

- `git push origin <branch>`

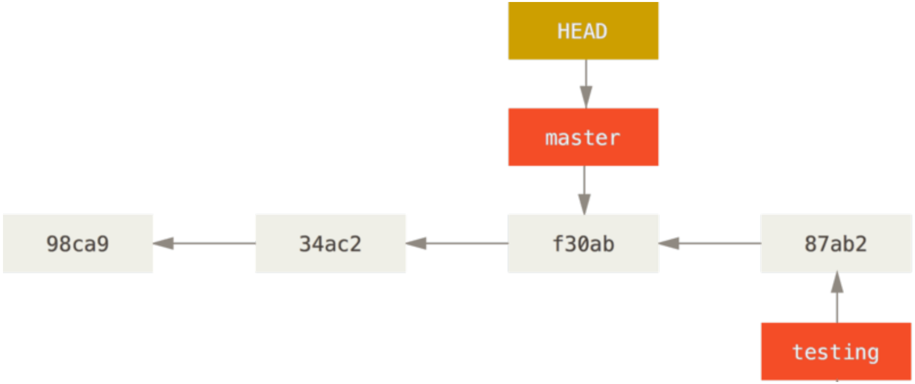
Switch between branches

- `git checkout <branch>`

Merge a branch into the current branch

- `git merge <branch>`

What's going on 2?



Git GUIs

Sorry, I was going to demo, I ran out of time. The following are cross-platform.

- Github Desktop
- Fork

Useful Resources

- Git - the simple guide
- The Git Community Book
- Git Cheatsheet