git 101

git

```
$ git
  - open source
  - distributed version control system
  - creation of Linus Torvalds
```

git config

```
$ # identity

$ git config --global user.name “John Smith”

$ git config --global user.email johnsmith@qut.edu.au

$ # editor

$ git config --global core.editor vim

$ # diff tool

$ git config --global merge.tool vimdiff
```

git config

```
$ # and because i like color

$ git config --global color.ui auto

$ # cross-platform development configuration

$ # for windows (note: if you are doing a windows-ONLY
  project set this to false)

$ git config --global core.autocrlf true

$ # for macosx/linux/unix/solaris/everything other than
  windows....

$ git config --global core.autocrlf input
```

git config

```
$ git config --list
  user.name=John Smith
  user.email=johnsmith@qut.edu.au
  color.ui=true
  core.editor=vim
  core.autocrlf=input
  merge.tool=vimdiff
```

git help

```
$ git help <verb>
```

```
$ git <verb> --help
```

```
$ man git-<verb>
```

git init

```
$ # make test directory

$ mkdir git-test-repo && cd git-test-repo

$ # initialising a repository

$ git init

$ # list created files/directories

$ ls -a (just `ls` for PowerShell)
  . .. .git

$ ls -a .git
  .    HEAD        config         hooks   objects
  ..   branches    description    info    refs
```

git status

```
$ # create file in repo

$ touch hello_world.c

$ # check the status of the repo

$ git status
On branch master

Initial commit

Untracked files:
  (use “git add <file>…” to include in what will be committed)

        hello_world.c

nothing added to commit but untracked files present (use “git add” to track)
```

git add

```
$ # track any changes to hello_world.c

$ git add hello_world.c

$ # check the status of the repo

$ git status
  On branch master

  Initial commit

  Changes to be committed:
    (use “git rm –cached <file>…” to upstage)

          new file:   hello_world.c
```

git commit

```
$ # commit the changes

$ git commit -m “initial commit”
  [master (root-commit) af9b016] initial commit
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 hello_world.c

$ # check the status of the repo

$ git status
  On branch master
  nothing to commit, working directory clean
```

“Commit Only Related Changes”

– Golden Rules of Version Control #1

to fork or to clone

```
$ # this be the question

$ # when to fork:
  - you want to make contributions to a open
  source project

$ # when to clone:
  - you want a local copy of the source code
  for, a) reading/editing etc b) compiling &
  running, but do NOT plan on contributing
  back to the project
```

github

```
$ # sign up to github, please

$ https://github.com/
```

fork a repo

```
$ # go to https://github.com/Lanzafame/qut-
  git-tutorial
```

git clone

```
$ # clone your recently forked repo

$ git clone https://github.com/yourUsername/qut-git-
  tutorial.git
  Cloning into ‘qut-git-tutorial’…
  remote: Counting objects: 5, done
  remote: Compressing objects: 100% (5/5), done.
  remote: Total 5 (delta 0), reused 0 (delta 0)
  Unpacking objects: 100% (5/5), done.
  Checking connectivity… done.

$ # change to cloned directory

$ cd qut-git-tutorial
```

git remote

```
$ git remote -v
  origin https://github.com/yourUsername/qut-git-tutorial.git (fetch)
  origin https://github.com/yourUsername/qut-git-tutorial.git (push)

$ # connect your local repo to the remote ‘master’ repo
```

```
$ git remote add upstream https://github.com/Lanzafame/qut-git-
  tutorial.git

$ git remote -v
  origin https://github.com/yourUsername/qut-git-tutorial.git (fetch)
  origin https://github.com/yourUsername/qut-git-tutorial.git (push)
  upstream https://github.com/Lanzafame/qut-git-tutorial.git (fetch)
  upstream https://github.com/Lanzafame/qut-git-tutorial.git (push)
```

git remote

```
$ # you can also add several remote repos
  # other than the upstream repo.
  # perfect application of this would be
  # team projects

$ # syntax: git remote add [alias] [repoURL]

$ git remote add lb https://github.com/LuBuss/
  qut-git-tutorial.git
```

git branch

```
$ # lets add a feature the ‘safe’ way

$ git branch hello_world
```

```
$ # list all our branches

$ git branch
   hello_world
  *master
```

git checkout

```
$ # change from master branch to hello_world
$ git checkout hello_world

$ # create file in repo
$ vim hello_world.c
```

hello_world.c

```c
#include <stdio.h>

int main (void) {
    printf("%s", "hello, world\n");
} // end function main
```

```
$ For the uninitiated:
  `i` -> to enter insert mode (editing)
  `ESC` -> to exit any mode back to normal/command mode
  `:w` -> to write/save a file
  `:q` -> to exit vi/m completely
  `:q!` -> to exit without saving
```

git diff

```
$ # lets stage hello_world.c

$ git add hello_world.c

$ # lets see the diff between staged and
  # committed files (in this case, an empty file)

$ git diff HEAD
  diff --git a/hello_world.c b/hello_world.c
  new file mode 100644
  index 0000000..f4cb5b
  --- /dev/null
  +++ b/hello_world.c
  @@ -0,0 +1,5 @@
  +#include <stdio.h>
  +
  +int main (void) {
  +       printf(“%s”, “hello, world\n”);
  +} // end function main
```

git commit

```
$ # let’s commit hello_world.c

$ git commit -m “added hello_world.c”
  [hello_world 7121aa4] Added hello_world.c
  1 file changed, 5 insertions(+)
  create mode 100644 hello_world.c

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

      hello_world

  nothing added to commit but untracked files present (use "git
  add" to track)
```

hello_world.c

```
$ # let’s add a line to hello_world.c
1 #include <stdio.h>
2
3 int main (void) {
4       printf(“%s”, “hello, world\n”);
5       printf(“%s”, “goodbye, world\n”);
5 } // end function main
```

git diff

```
$ # nows lets compare our latest changes to what
  # we have already staged

$ git diff
  diff --git a/hello_world.c b/hello_world.c
  index f4cb5b0..db851a3 100644
  --- a/hello world.c
  +++ b/hello_world.c
  @@ -2,4 +2,5 @@
   int main (void) {
           printf("%s", "hello, world\n");
  +        printf("%s", "goodbye, world\n");
  } // end function main

$ # understanding git diff output
  # http://www.git-tower.com/learn/ebook/command-line/
  advanced-topics/diffs
```

git checkout

```
$ # ok so let’s not say goodbye, but if you
  # are lazy (or careful) like me you don’t
  # want to edit the file directly so let’s
  # revert our changes to the last commit
```

```
$ git checkout -- hello_world.c

$ # we can either git status or git diff to
  # determine if it worked. git diff will
  # return blank. git status will return with
  # just untracked files.
```

git merge

```
$ # our feature is ready! let’s merge it back
  # back into the master branch

$ # first move back to the master branch

$ git checkout master
  Switched to branch ‘master'
  Your branch is up-to-date with 'origin/master'.

$ # now merge the hello_world branch

$ git merge hello_world
  Updating b659c24..7121aa4
  Fast-forward
    hello_world.c | 5 +++++
    1 file changed, 5 insertions(+)
    create mode 100644 hello_world.c
```

git fetch

```
$ # git fetch retrieves any changes from the remote repo but
  doesn’t merge them straight away

$ git fetch
  remote: Counting objects: 2, done.
  remote: Compressing objects: 100% (2/2), done.
  remote: Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
  Unpacking objects: 100% (2/2), done.
  From https://github.com/Lanzafame/qut-git-tutorial
     02fd872..40e498d  master     -> origin/master

$ git status
  On branch master
  Your branch is behind 'origin/master' by 1 commit, and can
  be fast-forwarded.
  ...
```

git merge

```
$ # now we have downloaded changes and we want to merge them because those
  files aren’t appearing in the directory yet

$ ls
  LICENSE       fetch.c       hello_world.c
  README.md     hello_world   hello_world.o
```

```
$ git merge origin
  Updating 02fd872..40e498d
  Fast-forward
   test.c | 0
   1 file changed, 0 insertions(+), 0 deletions(-)
   create mode 100644 test.c

$ ls
  LICENSE       fetch.c       hello_world.c
  README.md     hello_world   hello_world.o
  test.c
```

git pull

```
$ # the git pull command is a combination of
  the git fetch and git merge commands
  NOTE: this makes determining the cause of
  merge conflicts
```

git push

```
$ # git push pushes commits from your local repo to
  the remote repo

$ git push
  Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 333 bytes | 0 bytes/s,
done.
Total 3 (delta 1), reused 0 (delta 0)
To https://github.com/Lanzafame/qut-git-
tutorial.git
   40e498d..851b917  master -> master
```

git reset

```
$ Research this one yourself!
```

```
$ I don’t want to get blamed if you lose the
  last 5 hours of code because you used git
  reset based on this slide :)
```