|  |
| --- |
| SGS株式会社 |
| Gitマニュアル |
| EPARK developer Git Manual [Basic]  Git command description |

|  |
| --- |
| 作成：2014年3月10日 |

|  |  |  |
| --- | --- | --- |
| Revision date | Revised content | Revision of |
| 2014/03/10 | Create | 下平宏一 |
| 2014/04/16 | 4-6-3 the release branch update / 4-9 release after master. Add  Append command description git branch  Such typos, ☆ mark record changes | 下平宏一 |
| 2014/06/13 | Description at EPARK development time required and add to | 下平宏一 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

※Enter in the format of 2014 / 01 / 01 revised on the date.

Table of contents

[EPARK developer Git Manual [Basic] 3](#_Toc384745266)

[1.git And the？ 3](#_Toc384745267)

[2. Basic Git workflow 4](#_Toc384745268)

[3. Frequently-used commands 5](#_Toc384745269)

[4. Examples of various work 5](#_Toc384745270)

[4-1. Operation of the working environment 5](#_Toc384745271)

[4-2. Other operations such as branch release branch 6](#_Toc384745272)

[4-3. In other branches such as the release branch makes integration (merge) a branch of their own 6](#_Toc384745273)

[4-4. Git operation at the time of the work 7](#_Toc384745274)

[4-5. (If you want to change the work branch) to evacuate the work 7](#_Toc384745275)

[4-6. Preparing and updating at the time of its release 7](#_Toc384745276)

[4-7. Rebuilding the database 8](#_Toc384745277)

[4-8. Branch management of post-release 8](#_Toc384745278)

[4-9. Merge to the current working branch to master branch after release 8](#_Toc384745279)

[5. Anxiety and often-asked question 9](#_Toc384745280)

[Git Command Description 11](#_Toc384745281)

[git add 11](#_Toc384745282)

[git branch 11](#_Toc384745283)

[git checkout 12](#_Toc384745284)

[git commit 12](#_Toc384745285)

[git diff 13](#_Toc384745286)

[git fetch 13](#_Toc384745287)

[git log 13](#_Toc384745288)

[git merge 13](#_Toc384745289)

[git pull 14](#_Toc384745290)

[git push 14](#_Toc384745291)

[git remote 14](#_Toc384745292)

[git reset 14](#_Toc384745293)

[git revert 15](#_Toc384745294)

[git rm 15](#_Toc384745295)

[git stash 15](#_Toc384745296)

[git status 15](#_Toc384745297)

# EPARK Git documentation for developers [Fundamentals]

## git and the？

　Version control system "CVS" or "SVN (subversion)"

What is version control system?

A **version control system** (also known as a Revision Control System) is a repository of files, often the files for the source code of computer programs, with monitored access. Every change made to the source is tracked, along with who made the change, why they made it, and references to problems fixed, or enhancements introduced, by the change. It must have the the ability to track each change made to the repository.

I'm using a distributed version control system called Git in EPARK.

Features of Git is that it is possible to keep track of the change history of more than one environment.

Depending on the implementation of multiple environments, you can unit test to make development in the environment of the individual. After that, I will integrate the development has progressed to the white of the integration test environment.

White Environment

Schematic representation of multiple environment

Branch

0215-Release 0201-Release

0115\_Release

EPARK Production environment

You can create multiple branches in each environment.

Development environment of Mitsujiro's

EPARK Development environment of Taro

Branch

0201\_Release　Ticket\_5678

Jirou\_syusei

Branch

0201-Release　Ticket\_1234

Ticket\_9876 taro\_fix

The advantage of Git(EPARK),

　－Their work is not affected by changes in other

　－It performs the development in each environment, and does not affect to white environmental and production

　－You can run the unit tests to create a test data within their environment

　－Since the change of the individual is recorded in the remote repository, it is easy to be others to copy

　－In the individual circumstances, it is possible to restore from a remote repository such as source files even if lost

　The benefits of distributed and unique.

Just a little confusing at first, please learn the Git by reference to the following sites and this manual.

Git-related site of frequently used are as follows.

Wikipedia Git <http://ja.wikipedia.org/wiki/Git>

<http://kray.jp/blog/git-why-explanation/>

<http://www.backlog.jp/git-guide/>

<http://www.atmarkit.co.jp/ait/articles/1307/05/news028.html>

<http://www.hyuki.com/techinfo/gitinit.html>

<http://sourceforge.jp/git/>

First, let's examine the concept of command and Git

Software necessary to use　git

　－Terminal software such as　 **TeraTerm**

　－FTP Client program　 Such as　 **WinSCP**

　－File comparison and synchronization tools such as　 **WinMerge**

　－Git for Windows Git Client Tools

## 2. Git workflow of basic

**I recommend in the flow, such as the following.**

1. **I will create a branch.**

git branch　　4-1-1

1. **(For modification and addition of the source file)　I will work on branch.**

**3. I will stage the files that have been modified or added. (I will next time be committed)** git add　　4-5-1

**4. And commit.**

git commit　　4-5-2

**It is to get file that was staged, and changes are tracked to a remote repository.**

**5. I transferred to the remote repository to change the repository**

git push　　4-5-2

**It will flow like that.**

**I'll start by discussing the following command and flow required for each operation.**

Git workflow basic

Remote repository

**Program modification and addition**

**Work table-related**

**Staging**

**git add**

**Commit**

**git commit**

**Push**

**git push**

**Branching**

**git branch**

**Log**

**git status**

**git branch**

**git branch –-track**

**git pull**

Local repository under

## 

## 3. **Frequently used commands**

**Add : To the file specification in order to commit the changes (staging)**

**Branch : Branch create and view or delete branch information**

**Checkout : Switching and transfer of branch**

**Commit : After performing the add, recording Change log remote repositories**

**Diff : for display in diff format changes that have been made ​​to the file**

**Fetch : want to retrieve data from a remote repository**

**Grep : search for files**

**Log : see the commit log**

**Merge : By integrating the branch**

**Pull : To reflect changes in local repository from remote repository**

**Push : To reflect changes in remote repository form local repository**

**Stash : And temporarily stores the changes of work time**

**Status : Shows the files in which changes has been made**

## 4. Git Statements

### 4-1. Operation of the working environment

　　4-1-1. **Pull code from Master branch**

　　　A. Update current Branch from which they are derived (master)

　　　　1. **$ git checkout master** →Switching to the branch **master**

　　　　2. **$ git branch** →Confirmation of current branch

　　　　3. **$ git pull** →copy the modifications and changes of　master branch

　　　　Here Master branch (local) become the latest.

　　　B. Creating a development branch

※How to create a branch of name **phaseX-XXXX**.

　　　　1. $ **git branch phaseX-XXXX**　 →Creating a Branch (phaseX-XXXX)

　　　　2. $ **git push origin phaseX-XXXX –u**

　　　　　　　　　　→If want to add branch (phaseX-XXXX) to a remote repository.

　　　　　　　　　　　　※Local branch is related automatically.

Local repository

branch phasex-xxxx

branch master

Remote repository

**git pull**

branch master

copy the modifications and changes of remote to local

Branching

**git push -u**

branch phasex-xxxx

will send phasex-xxxx to remote

### 4-2. Other operations

　　C. How to create a local branch in the remote repository and release branch

　　　1. $ **git branch –r** →Confirmation of branch presence

　　　2. $ **git branch --track Branch Name origin / Branch Name**

→created locally from the remote repository

　　　3. $ **git branch** →Branch Name Confirmation

　　D. Update the branch in the environment

　　　1. $ **git checkout Branch name** →Switching of branch

　　　2. $ **git pull** → brings it up to date with the latest

### 4-3. integration (merge) a branch in other branch

　　E. Want to (merge) branch

　　　1. **$ git checkout Branch name of merge destination** → Switch to merge branch destination

　　　2. **$ git pull Branch name of the merge destination** → brings it up to date with the latest branch of the merge destination

　　　3. **$ git merge Branch name of the merge destination** → integrate to merge destination SOURCE

　Example

White Environment (release branch) Integrate branch A to (merge)

　　1. I will move to merge branch destination of your environment.

　　　$ git checkout 0101-release

　　2. Branch of the merge source (branch A) I want to merge to merge destination.

　　　$ git merge Branch　A

　　　‘A‘ branch has been merged into a destination for the merge.

　　　$ git push

　　　I reflect to remote information 0101-release merged

　　3. Login to white environment

　　4. cd epark

　　5. $ git branch → Confirmation of branch

　　　If the branch is different　checkout($ git checkout 0101-release) move to release branch

　　6. $ git pull → reflect the release of remote branch

　　Merge is now complete.

White environment (release branch) Integrate branch A to (merge)

Your environment

ブランチ

0101-releasee

ブランチ

0101-releasee

git push

ブランチ

A

git merge

Merge to 0101-release branch branch A

Remote

ブランチ

0101-releasee

git pull

White　Environment

　　F. Conflict check of merges

　　　1. Conflict happens if it is push or reject at merge time.

　　　　 Check the conflict by opening the file in WinMerge in question.

### 4-4. Git operation

　　4-4-1.File confirmation of changes and additions

**$ git status** → Shows the changes made file-by-file

**$ git add File name** → This is done by checking the files you have changed or added. (git add -A)

　　4-4-2. To commit

　　　By specifying all of the files in the changes in the add before commit

**$ git pull** → update branch.

**$ git commit** → enter the commit message in the commit run editor.

**$ git push** → will reflect all local changes.

### 4-5. (If you want to change the work branch) to save the current work

　　　1. $ **git stash save** → I save the current work

　　　2. $ **git checkout branch** → Switch to another branch, to perform the work

　　　3. Work restore it back to the original branch

　　　　 $ **git checkout Branch of the original**

　　　　 $ **git stash list**　　　　　 → Display and check the list that is in stash

　　　　 $ **git stash pop stash@{0}** → It will restore the stash saved at the stack{0}

### 4-6. Preparing and updating at the time of its release

　　4-6-1. **Merge branch**

　　　Branch A (merge destination phaseX-20140201-release)

　　　Branch B (merge original phaseX-XXXX-ticket number)

　　　＊Merge the contents of source (branch B) to the destination (branch A)

　　　1. $ **git checkout Branch　B** → Branch switching

　　　2. $ **git branch** → Branch name confirmation

　　　3. $ **git pull** → Get the latest status from a remote repository

　　　4. $ **git branch** → Confirmation of Branch are being created?

　　　　 If it has not been created, run the following command.

　　　　 git branch --track phase4-20140201-release origin/phase4-20140201-release

　　　5. $ **git branch** → Make sure that the branch A is displayed

　　　6. $ **git checkout ブランチA** → Switch to branch A

　　　7. $ **git pull** → reflect all the local repository

　　　8. $ **git merge ブランチB** → Merge branch of SOURCE

　　4-6-2. It reflects the work to the release branch

※It will push work to the release branch.

　　　1. **$ ./check-conflict.sh** → Conflict check

　　　If there is no conflict

　　　2. **$ git push** → I will push to a remote repository

　　　If there is a conflict

　　　I want to modify the source conflicts,

　　　3. **$ git commit –a** → To commit only modified files

　　　4. **$ git push** → I will push to a remote repository

　　4-6-3. (Pull on the release branch) update the release branch

　　　1. white Log in to the environment

　　　2. **$ git branch** → The Check is in release branch

　　　3. **$ git pull** → Pull from a remote repository

　※Release branch will be updated with this pull! !

### 4-7. Branch management post-release

　　4-7-1. **Delete the development branch**

**$** **git branch -d Branch name** →Delete local branch

　　　　→If you cannot remove

　　　　　 To forcibly remove the branch if there is a commit that has not been merged

**$ git branch -D branch name**

　　　If you want to delete a remote branch

To delete a branch from remote repository

**$ git push origin :branch\_name**

### 4-8. Merge master branch to the current working branch after release

It is a procedure that is performed when working with current working branch current work is to continue rather than release object.

　　　Latest branch of post-release (master) / current working branch (phase5-drugstore-XXXX)

　　　3. **$ git fetch** Get data from a remote repository

　　　4. **$ git checkout phase5-drugstore-XXX X** Switch to work branch

　　　5. **$ git branch** Confirmation of branch

　　　6. **$ git merge origin/master** Merge master branch to the current work

　　　7. If conflict message occurred at merge time-then try 8 to 10

　　　8. To perform a conflict check (WinMerge)

　　　9. I want to modify the source conflicts,

　　　10. **$ git push** It will push to a remote repository

## 5. FAQ

　　 1. I will commit your work

　　 　$ git add . Or $ git add --all

　　 　$ git commit -m 'Work' → To commit it with a name ('work')

　　 　$ git push → I want to push changes

　　 　Work file has been reflected in the remote repository now.

If you want to cancel from the commit the file that you add in 1.

　　 　$ git rm --cached File name → I cancel the file

　　 　$ git rm -r --cached Directory name → I want to cancel the directory

　　 　! ! rm command is dangerous, please command processing by specifying the file name.

　　 2. If you want to see a list of files that have changed

　　 　$ git status

　 A file that has been changed in are displayed. At the beginning of a line　modified: Files that have been changed、new file: It becomes the newly created file is.

　　 　$ git diff FILE-NAME

　　 　Ｒｅｄ It appears the changed parts of the are compared with before the change.

## 6. Git　Command Description

## git add

This command add the changes of working environment to the staging area.

However, only add commands does not reflect changes in the remote repository, it does not reflect the changes until you run the commit command.

How to use：

☆$ git add FILENAME

　　To commit the changes made ​​to the FILENAME.

☆git add DIRECTORY

　　To commit the changes in DIRECTORY.

☆git add --all or git add -A

　　It will be commit all files to what the changes have been made.

## git branch

This command is to perform create, list, rename or delete the branch.

How to use：

☆git branch

　　Show list of the local branch.

☆git branch -r

　　Show list of the branch in the remote repository.| grep can also be used.

☆git branch BRANCHname

　　Create a local branch named BRANCHname. Checkout to the new branch does not take place only new branch is created by this command.

☆git branch -d BRANCHname

　　Remove the local branch specified. Git command rejected if the changes that have not been merged into the branch if so then use the parameters of the –D inplace.

☆git branch -D BRANCHname

　　These commands forcibly remove even if there is a change that has not been merged.

☆git branch -m NEWNAME

　　It will change the name of the current branch to NEWNAME.

☆git branch -v

　　You can check the most recent commit of each branch.

☆git branch --merged

　　You can see list of branch merged into the branch you are.

☆git branch --no-merged

　　You can see the branch where there is work that has not been merged yet.

☆git branch -–track WORK-BRANCH origin/WORK-BRANCH

　　It is to create a branch named WORK-BRANCH locally with same name as WORK-BRANCH of the remote repository.

## git checkout

This command has three functions checkout the branch (switching and transfer), checkout the commit, and checkout the file.

When we cannot checkout an error message is displayed when there is a file of work in progress (those not staged). In that case we can add → commit, checkout will not run unless you temporarily saved using the stash.

How to use：

☆git checkout phaseX-epark-1234

　　This will switch to phaseX-epark-1234 branch.

git checkout 85c61613211ba2be9af80fd1e6107b23d289e3a8

　　It is a command to update the state of the commit using commit hash of 85c · · ·.

　　※A hash attached to 85c · · · at commit time. You can check the hash in git log.

☆git checkout BRANCH\_NAME

　　Swith to the branch BRANCH\_NAME

　git checkout -b NEWBRANCH

　　It is a command that creates a new branch NEWBRANCH, to check out at the same time.

　git checkout -f

　　Check out and undo all changes made during work.

　　※All changes will be lost.

☆git checkout FILENAME

　　undo the changes in the file.

　　※Changes are lost when you run this command, to return to the previous change.

## git commit

It is a command to commit the changes to the remote repository file.

How to use：

☆git commit

　Text editor is invoked when we run this command here we can add name to commit.

☆git commit -m 'NAME'

　It is a parameter that add name to the commit without having to start the text editor. Commit is performed NAME enclosed in single quotes is the name.

　　※Always add ticket number to the commit name.

☆git commit -a

　　It will commit all changes in the working environment.

　git commit --amend

　　The command is useful if you want to modify the last commit.

## git diff

This is the command to see the changes.

How to use：

　git diff

　　Run with no parameters if you see what you have changed, but it is not yet staged.

　git diff --staged

　　If you want to see what is supposed to be committed.

☆git diff FILENAME.php

　　You specify the file name in order to see the detailed changes in FILENAME.php.

## git fetch

**This command gets what local does not have out of all the data in the remote.**

How to use：

　git fetch

　　This command will only get from a remote repository. Merge does not perform.

## git log

Use this command to display a list of committed.

How to use：

☆git log

　This defaults and format all commit history. Enter the q In the case of the end.

　git log -n LIMIT

　It will limit the number of histories to be displayed.

　git log -p

　Appears (diff) diffs corresponding to commit 'commit all'.

　git log -–author=”USER-NAME”

　Display commits that contain "USER-NAME". It often used to check post-commit.

## git merge

This command is to merge one branch into other branch.

How to use：

☆git merge BRANCH-A

　　This command is to merge the specified branch (BRANCH-A) into current branch.

## git pull

This command is to merge into the local repository the change made to remote repository.

How to use：

☆git pull

　　It will merge the change of the remote repository to the current branch.

It becomes the same operation as running continuously the merge command and fetch command.

## git push

This will merge the changes that has been made in the local repository to a remote repository.

How to use：

☆git push

　　It will send the change of local repository to the remote repository.

☆git push origin : Remote branch name

　　It will push changes to the remote destination branch.

## git reset

It is a command used to cancel the commit that you specify.

How to use：

　git reset –-hard HEAD

　　To match the state of the commit just before the reset and working environment. - hard is a parameter to undo all changes in the work environment.

git reset 85c3a0e871ff60411ca89fa07c7f2b4d426fa04285d

　　It is the command to return to the commit you specify. Work environment will still be saved without returning to the original.

　git reset --hard 85c3a0e871ff60411ca89fa07c7f2b4d426fa04285d

　　Back to the commit you specify, to return to the state.

◇Individual file commit(Example of git reset)

Run the git add by changing the b.php a.php and work environment.

git add a.php

git add b.php

At this rate, how to create a commit that you have changed only b.php is as follows:

git reset a.php It will cancel the add of a.php in this state.

git commit-m 'change to b.php' b.php is committed

I commit to add the a.php again

git add a.php

changes git commit-m 'edit to a.php' a.php is committed

## git revert

It is a command to undo the changes that have been committed.

How to use：

　git revert 85c85c61613211ba2be9af80fd1e6107b23d289e3a8

　Message to name the commit will be displayed when you run this command. Commit is created when you save it by entering the message.

Example messages this reverts commit 3a0e871ff60411ca89fa07c7f2b4d426fa04285d

## git rm

It is a command that you can delete only file information and committed staging, delete the file itself.

How to use：

　　git rm –cached FILENAME.EXTENSION →Delete only information that has been committed, leaving the file

　　git rm FILENAME.EXTENSION →Information that you have committed both of files to be deleted

## git stash

This command is used when to retracted temporarily work.

Use this when you want to move to another branch when you can not commit at work.

How to use：

☆ git stash save

It save the status of the work.

git stash list

Show me a list of work temporarily saved.

git stash clear

It will remove all the work that has been temporarily saved.

## git status

Use this command to display the status of the file to be committed.

Files that have been newly created file, modified file.

How to use：

☆git status

　　Shows a list of files that have been created and modified file.

Modified file: ※RED

Staged file: ※Green

Untracked files: the newly created file that does not exist in the previous commit (tracked out = untrack) ※RED

new file: ※Green

## Git commands and there use

* git log -- displays the record activity of a branch
* git branch -- displays the list of branch within the folder (the branch colored with green is that branch in which we are currently present).
* git status -- shows the status of the branch in which we are currently present. It shows the list of file in which changes has been made (i.e. it shows the list of modified file in local repository that is different from the remote repository).
* git add test-senjibu -- add the branch update to the Head.
* git branch test-senjibu -- creates a new branch of name as passed.
* git push origin test-senjibu -- this will push the new branch to the origin (i.e. remote repository)
* git branch -r | grep senjibu -- this will search the branch of remote repository filtering by name.
* git branch -d test-senjibu -- this will delete the branch as passed by name.
* git branch --track branchName origin/branchName -- Branch branchName set up to track remote branch branchName from origin.
* git checkout branchName -- this will Switched to branch ' branchName '
* ./symfony cc -- This commands clears the cache. After any pull and push command to see the changes in the user mode we run this command.
* symfony-propel-build-model-forms-filters-sql.php -- if we make any changes to the schema or table then we push it in the master to reflect the changes, after that we run the command symfony-propel-build-model-forms-filters-sql.php

**To Push changes to a branch**

1. git status
2. git pull
3. git add
4. git commit –m “some work on makecheckbox.php”
5. git push origin branchName

## Git Examples

### Create new branch

git branch testing

To switch to branch you have to checkout the branch

git checkout testing

### Rename a branch

git branch -m [old\_name] [new\_name]

### Delete a branch

*# delete branch testing*

git branch -d testing

*# force delete testing*

git branch -D testing

*# check if branch has been deleted*

git branch

## Revert uncommitted changes

*# delete a file*

**rm** test01

*# revert the deletion*

git checkout -- test01

### Remove staged changes for new files

If you added a new file to the staging area but do not want to commit the file, you can remove it from the staging area via the  git reset file  command.

Example - to remove incorrect.txt

git add .

*# remove it from the staging area*

git reset incorrect.txt

*# to clean up, delete the file*

**rm** incorrect.txt

### Checkout committed versions

You can check out older revisions of your file via the commit ID.

git log

*# checkout the older revision via commit\_id*

git checkout [commit\_id]

*# based on the example output this could be*

git checkout 046474a52e0ba1f1435ad285eae0d8ef19d529bf

### Reverting a commit

*# revert a commit*

git revert commit\_id

### Setting up tracking branches

If you clone a Git repository, your local master branch is created as a tracking branch for the master branch of the origin repository (short: origin/master) by Git.

You create new tracking branches by specifying the remote branch during the creation of a branch. The following example demonstrates that.

*# setup a tracking branch called newbrach*

*# which tracks origin/newbranch*

git checkout -b newbranch origin/newbranch

### Updating your remote tracking branches with git fetch

git fetch origin

The git fetch command updates your remote tracking branches, i.e. it updates the local copy of branches stored in a remote repository. The following command updates the remote tracking branches from the repository called origin.

### Merging

Git allows you to combine the changes of two branches. This process is called merging.

*# merges into your currently checked out branch*

git merge testing

### Fetch compared with pull

The git pull command performs a git fetch and git merge. The git fetch does not perform any operations on your local branches. You can always run the fetch command and review the incoming changes.

### Solving merge conflicts

A merge conflict occurs if two commits from different branches have modified the same content and Git cannot automatically determine how both changes should be combined when merging these branches.

If a merge conflict occurs, Git will mark the conflict in the file and the programmer has to resolve the conflict manually. After resolving it, he can add the file to the staging area and commit the change.

### Common Git Commands

Git checkout HEAD /filepath

* To remove changes made by me

To undo the last commit

* Git reset -–soft HEAD~1

Git diff /filepath

* To see the changes

To commit changes (To development branch)

* Git pull
* Git status
* Git add –A
* Git commit //always write ticket number in commit comment
* Git push -n origin branch-name //this is dry run to check for any errors but this will not push
* Git push origin branch-name //this is final push
* Git log

To merge content to another branch (from development branch to release branch)

* Git checkout phase5-release
* Git pull origin phase5-release
* Git merge phase5-development
* Git push origin phase5-release

To track new development branch

- git checkout master

- git pull //update master

- git branch –r //check remote branches

- git branch –-track branchName origin/branchName //track the branch which we want to merge in local

- git branch // check track branch is visible in local

- git checkout branchName //switch to track branch

- git pull origin branchName //pull the latest code to local repository

- git log // check for recent changes in log to confirm if code updated

To merge content to another branch (from development branch to release branch)

Git pull

Git branch –r //check release branch in remote repository

git branch --track phase5-1009-release origin/phase5-1009-release //track release branch

git checkout phase5-1009-release //checkout to release branch

git branch

git pull origin phase5-1009-release //pull the origin code in release branch

git log

git branch

git merge phase5-membership-14334 //merge the content of development branch to release branch

git push origin phase5-1009-release //push the content of release branch in remote sever

git log

Command after creating a new branch (after release)

* Git checkout master
* Git pull origin master
* Git log
* Git branch
* Git checkout newBranch
* Git merge master //After creating a branch merge the code of master in local branch
* Git log

To undo last commit

* Git reset –-soft HEAD~1

After including branch in Local repository

* Git pull origin branch\_name
* Git merge master

To check last push made by me-

git push -n origin phase5-jobscheduler-test

git log -5

To pull new code after master updated

Git checkout master

Git pull

To get the name of files which has been modified in a commit

* git diff a35fc6fdc658d5b355e210420376cd87311b623d^ a35fc6fdc658d5b355e210420376cd87311b623d --name-only

To create branch in remote repository

* git checkout master
* git branch
* git pull
* git branch phaseX-XXXXX
* git push origin phaseX-XXXXX –u

To delete branch from remote repository

* git push origin :branch\_name

**To check who made changes to the file**

* git blame file\_name

This will show line by line who made what changes to the file.

To solve conflict

-open the conflict file

<<<<<<< HEAD

<link type="text/css" rel="stylesheet" media="all" href="style.css" />

=======

<!-- no style -->

>>>>>>> master

* The conflict has happened between
* The first HEAD section is the version of the current branch (style) head. The second MASTER section is the version of master branch.
* Solve the conflict manually by making proper changes to the code
* Add file
* git add lib/hello.html
* commit file
* git commit -m "Merged master fixed conflict.

A. Update current Branch from which they are derived (master)

　　　　1. **$ git checkout master** →Switching to the branch **master**

　　　　2. **$ git branch** →Confirmation of current branch

　　　　3. **$ git pull** →copy the modifications and changes of　master branch

　　　　Here Master branch (local) become the latest.

B. Creating a development branch

※How to create a branch of name **phaseX-XXXX**.(checkout master before creating a new branch).

　　　　1. $ **git branch phaseX-XXXX**　 →Creating a Branch (phaseX-XXXX)

　　　　2. $ **git push origin phaseX-XXXX –u**

　　　　　　　　　　→If want to add branch (phaseX-XXXX) to a remote repository.

　　　　　　　　　　　　※Local branch is related automatically.

4-7-1. **Delete the development branch**

**$** **git branch -d branch\_name** → Delete local branch

　　　　→ If you cannot remove

　　　　　 To forcibly remove the branch if there is a commit that has not been merged

**$ git branch -D branch\_name**

　　　If you want to delete a remote branch

To delete a branch from remote repository

**$ git push origin :branch\_name**

**To check who made what changes to the file**

* git blame file\_name

This will show line by line who made what changes to the file.

* git log –p file\_name

this will show the previous changes made on file (git blame only shows the newly added and the edited part of a file this command is useful if we want to see the deleted part too).

**Newly learned commands**

1. git log --name-status – this command will display the logs along with the username of the person who commited and the list of files that has been commited.
2. Vim editor – benefits over vi is that

* It includes multilevel undo/redo.
* Vim allows screen to be split for editing multiple files
* Vim includes a built in diff for comparing files (vimdiff)